ISO TC 184/SC4/ ___ Nxxxx **Date:** 2000/09/11

Supersedes ISO TC 184/SC4/ __ N ___

ISO/WD 10303-2fd
Product data representation and exchange: Application protocol: Fluid dynamics data

ABSTRACT:
This provides an initial draft of the AP for fluid dynamics data.

KEYWORDS: Computational fluid dynamics, Wind tunnel test, Flight test

COMMENTS TO READER:
The formal modeling uses EXPRESS Amendment 1.This version has revised ARM clauses with respect to the prior version dated 2000/08/08. Changes for for the August 2000 SIDS are included but the AIM is in a state of disrepair.

Project Leader: Ray Cosner
Address: Boeing, Phantom Works
PO Box 516,
M/S S106–7126
St. Louis, MO 63166

Telephone: +1 (314) 233–6481
Telefacsimile: +1 (314) 777–1328
Electronic mail: raymond.r.cosner@boeing.com

Project Editor: Peter Wilson
Address: Boeing Commercial Airplane
PO Box 3707, M/S 6H–AF
Seattle, WA 98124-2207

Telephone: +1 (425) 237–3506
Telefacsimile: +1 (425) 327–3428
Electronic mail: peter.r.wilson@boeing.com

International Organization for Standardization
Case Postale 56 • CH–2111 Genève 20 • Switzerland

Contents

Page

1 Scope . . . 1

2 Normative references . . . 3

3 Terms, definitions, abbreviations, and symbols . . . 3
3.1 Terms defined in ISO 10303-1 . . . 3
3.2 Other definitions . . . 3
3.3 Abbreviations . . . 4
3.4 Symbols . . . 4

4 Information requirements . . . 7
4.1 Units of functionality . . . 7
4.1.1 UoF1 . . . 7
4.2 Application objects . . . 7
4.3 Application assertions . . . 7

5 Application interpreted model . . . 8
5.1 Mapping table . . . 8
5.2 AIM EXPRESS short listing . . . 10
5.2.1 Fundamental concepts and assumptions . . . 12
5.2.2 Fluid dynamics data types . . . 12
5.2.2.1 Fluid dynamics data type definitions . . . 12
5.2.2.1.1 cfd_standard_data_name . . . 12
5.2.2.1.2 flow_solution_data_name . . . 12
5.2.2.1.3 turbulence_data_name . . . 15
5.2.2.1.4 nondimensional_data_name . . . 16
5.2.2.1.5 Riemann_1D_data_name . . . 17
5.2.2.1.6 force_moment_data_name . . . 19
5.2.2.1.7 bc_type . . . 21
5.2.2.1.8 bc_type_simple . . . 21
5.2.2.1.9 bc_type_compound . . . 24
5.2.2.1.10 governing_equations_type . . . 24
5.2.2.1.11 gas_model_type . . . 25
5.2.2.1.12 viscosity_model_type . . . 26
5.2.2.1.13 thermal_conductivity_model_type . . . 27
5.2.2.1.14 turbulence_closure_type . . . 28
5.2.2.1.15 turbulence_model_type . . . 29
5.2.2.2 Fluid dynamics data imported type definitions . . . 30
5.2.2.2.1 TYPE 1 . . . 30
5.2.2.3 Fluid dynamics data entity definitions . . . 30
5.2.2.3.1 cfd_base . . . 30
5.2.2.3.2 zone . . . 31
5.2.2.3.3 structured_zone . . . 33
5.2.2.3.4 unstructured_zone . . . 34
5.2.2.3.5 grid_coordinates . . . 34

5.2.2.3.6 flow_solution . . . 36
5.2.2.3.7 zone_bc . . . 37
5.2.2.3.8 bc . . . 38
5.2.2.3.9 bc_data_set . . . 42
5.2.2.3.10 bc_data . . . 46
5.2.2.3.11 family . . . 48
5.2.2.3.12 flow_equation_set . . . 48
5.2.2.3.13 governing_equations . . . 50
5.2.2.3.14 fd_model . . . 51
5.2.2.3.15 gas_model . . . 51
5.2.2.3.16 thermal_conductivity_model . . . 52
5.2.2.3.17 turbulence_closure . . . 53
5.2.2.3.18 turbulence_model . . . 54
5.2.2.3.19 viscosity_model . . . 55
5.2.2.3.20 reference_state . . . 56
5.2.2.3.21 convergence_history . . . 57
5.2.2.3.22 discrete_data . . . 59
5.2.2.3.23 integral_data . . . 60
5.2.2.4 Fluid dynamics data function definitions . . . 62
5.2.2.4.1 schname . . . 62
5.2.2.4.2 schdot . . . 62
5.2.2.4.3 consistent_data_array . . . 62
5.2.2.4.4 inherit_class_from_base_zone . . . 63
5.2.2.4.5 inherit_units_from_base_zone . . . 64
5.2.2.4.6 inherited_class_for_refstate . . . 65
5.2.2.4.7 inherited_units_for_refstate . . . 65
5.2.2.4.8 grid_data_size . . . 66
5.2.2.4.9 field_data_size . . . 67

6 Conformance requirements . . . 69

Annex A (normative) AIM EXPRESS expanded listing . . . 70

Annex B (normative) AIM short names . . . 71

Annex C (normative) Implementation method specific requirements . . . 72

Annex D (normative) Protocol Implementation Conformance Statement (PICS) proforma 73

Annex E (normative) Information object registration . . . 74

Annex F (informative) Application activity model . . . 75
F.1 Introduction . . . 75
F.2 Application activity model background . . . 75
F.3 Application activity model definitions . . . 77
F.4 Application Process Description . . . 82
F.5 Process Variant Approaches . . . 90

F.5.1 Equations of Fluid Dynamics . . . 90
F.5.2 Computational Grid . . . 91
F.5.3 Flow Physics Models . . . 92

Annex G (informative) Application reference model . . . 93
G.1 Introduction . . . 93
G.2 hierarchy . . . 94
G.2.1 Introduction . . . 95
G.2.2 Fundamental concepts and assumptions . . . 95
G.2.3 hierarchy entity definitions . . . 96
G.2.3.1 analysis . . . 96
G.2.3.2 cfd_case . . . 99
G.2.3.3 product_analysis . . . 100
G.2.3.4 product . . . 101
G.2.3.5 zone . . . 101
G.2.3.6 structured_zone . . . 103
G.2.3.7 unstructured_zone . . . 104
G.2.3.8 element . . . 104
G.2.4 hierarchy function definitions . . . 105
G.2.4.1 derive_zone_dimension . . . 105
G.3 basis . . . 105
G.3.1 Introduction . . . 106
G.3.2 Fundamental concepts and assumptions . . . 106
G.3.3 basis type definitions . . . 107
G.3.3.1 label . . . 107
G.3.3.2 textual . . . 108
G.3.3.3 data_class . . . 109
G.3.3.4 mass_units . . . 111
G.3.3.5 length_units . . . 111
G.3.3.6 time_units . . . 111
G.3.3.7 temperature_units . . . 111
G.3.3.8 angle_units . . . 112
G.3.3.9 grid_location . . . 112
G.3.3.10 data_name . . . 112
G.3.3.11 adhoc_data_name . . . 113
G.3.3.12 standard_data_name . . . 113
G.3.3.13 coordinate_data_name . . . 113
G.3.3.14 nondimensional_data_name . . . 114
G.3.4 basis entity definitions . . . 115
G.3.4.1 cfd_pdm . . . 115
G.3.4.2 array_of_data . . . 115
G.3.4.3 data_conversion . . . 115
G.3.4.4 dimensional_units . . . 116
G.3.4.5 dimensional_exponents . . . 117
G.3.4.6 index_list . . . 117
G.3.4.7 index_range . . . 118

G.3.4.8 data_array . . . 118
G.3.4.9 dimensional_data_array . . . 120
G.3.4.10 nondimensional_data_array . . . 120
G.3.4.11 rind . . . 121
G.3.4.12 geometry_reference . . . 121
G.4 domain . . . 122
G.4.1 Introduction . . . 122
G.4.2 Fundamental concepts and assumptions . . . 122
G.4.3 domain entity definitions . . . 125
G.4.3.1 grid_coordinates . . . 125
G.4.3.2 zone_grid_connectivity . . . 127
G.4.3.3 connectivity . . . 128
G.5 conditions . . . 129
G.5.1 Introduction . . . 129
G.5.2 Fundamental concepts and assumptions . . . 130
G.5.3 conditions type definitions . . . 132
G.5.3.1 Riemann_1D_data_name . . . 132
G.5.3.2 bc_type . . . 139
G.5.3.3 bc_type_simple . . . 140
G.5.3.4 bc_type_compound . . . 143
G.5.4 conditions entity definitions . . . 143
G.5.4.1 condition . . . 143
G.5.4.2 zone_bc . . . 144
G.5.4.3 bc . . . 145
G.5.4.4 family . . . 148
G.5.4.5 bc_data_set . . . 149
G.5.4.6 bc_data . . . 153
G.5.4.7 reference_state . . . 154
G.5.4.8 integral_data . . . 156
G.5.5 conditions function definitions . . . 157
G.5.5.1 bcdataset_data_array_length . . . 157
G.5.5.2 bcdata_class . . . 157
G.5.5.3 bcdata_dimunits . . . 158
G.5.5.4 inherit_class_from_base_zone . . . 159
G.5.5.5 inherit_units_from_base_zone . . . 159
G.5.5.6 inherited_class_for_refstate . . . 160
G.5.5.7 inherited_units_for_refstate . . . 161
G.6 equations . . . 162
G.6.1 Introduction . . . 162
G.6.2 Fundamental concepts and assumptions . . . 162
G.6.3 equations type definitions . . . 162
G.6.3.1 turbulence_data_name . . . 162
G.6.3.2 force_moment_data_name . . . 163
G.6.3.3 governing_equations_type . . . 169
G.6.3.4 gas_model_type . . . 170
G.6.3.5 viscosity_model_type . . . 171

G.6.3.6 thermal_conductivity_model_type . . . 172
G.6.3.7 turbulence_closure_type . . . 173
G.6.3.8 turbulence_model_type . . . 174
G.6.4 equations entity definitions . . . 175
G.6.4.1 equation . . . 175
G.6.4.2 flow_equation_set . . . 175
G.6.4.3 governing_equations . . . 177
G.6.4.4 fd_model . . . 178
G.6.4.5 gas_model . . . 179
G.6.4.6 thermal_conductivity_model . . . 179
G.6.4.7 turbulence_closure . . . 180
G.6.4.8 turbulence_model . . . 181
G.6.4.9 viscosity_model . . . 181
G.7 results . . . 182
G.7.1 Introduction . . . 183
G.7.2 Fundamental concepts and assumptions . . . 183
G.7.3 results type definitions . . . 183
G.7.3.1 flow_solution_data_name . . . 183
G.7.4 results entity definitions . . . 187
G.7.4.1 result . . . 187
G.7.4.2 flow_solution . . . 188
G.7.4.3 convergence_history . . . 189
G.7.4.4 discrete_data . . . 190

Annex H (informative) AIM EXPRESS-G . . . 193

Annex J (informative) AIM EXPRESS listing . . . 194

Annex K (informative) Application protocol usage guide . . . 195

Annex L (informative) Technical discussions . . . 196

Bibliography . . . 197

Index . . . 198

Figures

F.1 A0 Design, build, test and ship product . . . 83
F.2 A0 Design, build, test and ship product . . . 84
F.3 A1 Design product . . . 85
F.4 A12 Conduct analysis of design . . . 86
F.5 A121 Build and run analysis model . . . 87
F.6 A122 Extract results . . . 88
F.7 Data flow through the CFD process . . . 89
G.1 Topologically based CFD hierarchy . . . 94
G.2 Schema level ARM diagram (page 1 of 1) . . . 95

G.3 Entity level diagram of ARM hierarchy schema (page 1 of 4) . . . 96
G.4 Entity level diagram of ARM hierarchy schema (page 2 of 4) . . . 97
G.5 Entity level diagram of ARM hierarchy schema (page 3 of 4) . . . 98
G.6 Entity level diagram of ARM hierarchy schema (page 4 of 4) . . . 99
G.7 Entity level diagram of ARM basis schema (page 1 of 5) . . . 107
G.8 Entity level diagram of ARM basis schema (page 2 of 5) . . . 108
G.9 Entity level diagram of ARM basis schema (page 3 of 5) . . . 109
G.10 Entity level diagram of ARM basis schema (page 4 of 5) . . . 110
G.11 Entity level diagram of ARM basis schema (page 5 of 5) . . . 110
G.12 Example convention for a 2-D cell center . . . 123
G.13 Example grid block with rind vertices . . . 123
G.14 A 1-to-1 abutting interface . . . 124
G.15 A mismatched abutting interface . . . 124
G.16 An overset interface . . . 125
G.17 Entity level diagram of ARM domain schema (page 1 of 2) . . . 126
G.18 Entity level diagram of ARM domain schema (page 2 of 2) . . . 126
G.19 Hierarchy for boundary-condition structures . . . 131
G.20 Entity level diagram of ARM conditions schema (page 1 of 8) . . . 132
G.21 Entity level diagram of ARM conditions schema (page 2 of 8) . . . 133
G.22 Entity level diagram of ARM conditions schema (page 3 of 8) . . . 134
G.23 Entity level diagram of ARM conditions schema (page 4 of 8) . . . 135
G.24 Entity level diagram of ARM conditions schema (page 5 of 8) . . . 136
G.25 Entity level diagram of ARM conditions schema (page 6 of 8) . . . 137
G.26 Entity level diagram of ARM conditions schema (page 7 of 8) . . . 138
G.27 Entity level diagram of ARM conditions schema (page 8 of 8) . . . 139
G.28 Entity level diagram of ARM equations schema (page 1 of 10) . . . 163
G.29 Entity level diagram of ARM equations schema (page 2 of 10) . . . 163
G.30 Entity level diagram of ARM equations schema (page 3 of 10) . . . 164
G.31 Entity level diagram of ARM equations schema (page 4 of 10) . . . 165
G.32 Entity level diagram of ARM equations schema (page 5 of 10) . . . 166
G.33 Entity level diagram of ARM equations schema (page 6 of 10) . . . 166
G.34 Entity level diagram of ARM equations schema (page 7 of 10) . . . 167
G.35 Entity level diagram of ARM equations schema (page 8 of 10) . . . 167
G.36 Entity level diagram of ARM equations schema (page 9 of 10) . . . 168
G.37 Entity level diagram of ARM equations schema (page 10 of 10) . . . 169
G.38 Entity level diagram of ARM results schema (page 1 of 4) . . . 183
G.39 Entity level diagram of ARM results schema (page 2 of 4) . . . 184
G.40 Entity level diagram of ARM results schema (page 3 of 4) . . . 185
G.41 Entity level diagram of ARM results schema (page 4 of 4) . . . 186

Tables

1 Symbols for dimensional units . . . 4
2 Symbols for coordinate systems . . . 4
3 Symbols for unit vectors . . . 5
4 Symbols for physical properties . . . 5

5 Symbols for nondimensional parameters and related scales 6
6 Flow solution data name identifiers 14
7 Turbulence data name identifiers 16
8 Nondimensional data name identifiers 18
9 Riemann 1-D data name identifiers 18
10 Force and moment data name identifiers 20
11 InwardNormalIndex values 40
12 Associated boundary-condition types and usage rules 43
G.1 inward_normal_index values 147
G.2 Associated boundary-condition types and usage rules 150
G.3 Data that may be associated with **reference_state** 155
G.4 Encoding of the 3-D **diffusion_model** 178

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75% of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

International Standard ISO 10303–2fd was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC4, *Industrial data*.

This International Standard is organized as a series of parts, each published separately. The parts of ISO 10303 fall into one of the following series: description methods, integrated resources, application interpreted constructs, application protocols, abstract test suites, implementation methods, and conformance methods. The series are described in ISO 10301-1.

A complete list of parts of ISO 10303 is available from the Internet:
<http://www.nist.gov/sc4/editing/step/titles/>

This part of ISO 10303 is a member of the application protocol series.

Annexes A, B, C, D and E are a normative part of this International Standard. Annexes G, H, J, K and L are for information only.

Introduction

ISO 10303 is an International Standard for the computer-interpretable representation and exchange of product data. The objective is to provide a neutral mechanism capable of describing product data throughout the life cycle of a product independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving.

Major subdivisions of this International Standard are:

— TBD

Application protocols provide the basis for developing implementations of ISO 10303 and abstract test suites for the conformance testing of AP implementations.

Clause 1 defines the scope of the application protocol and summarizes the functionality and data covered by the AP. Clause 3 lists the words defined in this part of ISO 10303 and gives pointers to words defined elsewhere. An application activity model that is the basis for the definition of the scope is provided in annex F. The information requirements of the application are specified in clause 4 using terminology appropriate to the application. A graphical representation of the information requirements, referred to as the application reference model, is given in annex G.

Resource constructs are interpreted to meet the information requirements. This interpretation produces the application interpreted model (AIM). This interpretation, given in 5.1, shows the correspondence between the information requirements and the AIM. The short listing of the AIM specifies the interface to the integrated resources and is given in 5.2. Note that the definitions and EXPRESS provided in the the integrated resources for constructs used in the AIM may include select list items and subtypes which are not imported into the AIM. The expanded listing given in annex A contains the complete EXPRESS for the AIM without annotation. A graphical representation of the AIM is given in annex H. Additional requirements for specific implementation methods are given in annex C.

In this International Standard the same English language words may be used to refer to an object in the real world or to a concept, and as the name of an EXPRESS data type that represents this object or concept. The following typographical convention is used to distinguish between these. If a word or phrase occurs in the same typeface as narrative text, the referent is the object or concept. If the word or phrase occurs in a bold typeface, the referent is the EXPRESS data type. Names of EXPRESS schemas also occur in a bold typeface.

The name of an EXPRESS data type may be used to refer to the data type itself, or to an instance of the data type. The disctinction between these uses is normally clear from the context. If there is a likelihood of ambiguity, the phrase ‘entity data type’ or ‘instance(s) of’ is included in the text.

Quotation marks “ ” are used to denote text that is copied from another document. Inverted commas ‘’ are used to denote particular string values.

Several components of this part of ISO 10303 are available in electronic form. This access is provided through the specification of Universal Resource Locators (URLs) that identify the location of these files on the Internet. If there is difficulty accessing these sites contact the ISO Central Secretariat or the ISO TC184/SC4 Secretariat directly at: `sc4@cme.nist.gov`.

Industrial automation systems and integration — Product data representation and exchange — Part 2fd : Application protocol: Fluid dynamics data

1 Scope

This part of ISO 10303 specifies the use of the integrated resources necessary for the scope and information requirements for the explicit representation of computer-readable data in the discipline of fluid dynamics. This standard will be applicable to all industries requiring representation of a fluid dynamic flowfield, including aerospace, automotive, and shipbuilding industries.

NOTE The application activity model in annex F provides a graphical representation of the processes and information flows which are the basis for the definition of the scope of this part of ISO 10303.

The following are within the scope of this part of ISO 10303:

— digital data on structured and unstructured grids describing steady or unsteady fluid dynamics flowfields;

— data describing the fluid dynamics model including grid description, grid inter-connectivity, boundary conditions, and modeling parameters;

— data from solutions of equation sets commonly used in fluid dynamics analysis: Navier-Stokes equations, Euler equations, linear and nonlinear potential flow equations, small-disturbance equations, boundary layer equations, and stream function equations;

— single-phase flow of a liquid or a gas;

— laminar flow, transitional flow, turbulent flow (direct representation of turbulence, or represented by Reynolds-averaged data);

— incompressible or compressible flow;

— unsteady flow;

— perfect gas, or variable chemical composition (equilibrium flow, frozen flow, or finite-rate chemical reactions);

— data regarding the exchange of energy by molecular transport including convection, conduction, and advectation;

— rotating flowfields (e.g., turbomachinery);

— inertial and rotating frames of reference;

— Newtonian transport laws;

— reference to product geometry;

— administrative information necessary to track the approval and configuration control of the analysis of a product;

The following are outside the scope of this part of ISO 10303:

— representations of geometry;

— gross flow in networks (e.g., piping and ducting);

— the use that application programs may make of the data;

— the means by which application programs modify the data;

— the form in which the data is stored internal to an application.

The validity, accuracy and completeness of the data for a particular purpose are determined entirely by the applications' software.

NOTE 1 The following are outside the scope of this edition of this part of ISO 10303 but are expected to be inside the scopes of later editions of this part:

— two- and three-phase flow;

— free surface flow;

— non-continuum flow (e.g., direct simulation of Monte Carlo data);

— data from non-analytical sources (e.g., experimental simulation such as wind tunnel or water tank testing, and product test such as flight test or sea trials);

— data regarding the exchange of energy by radiation;

— non-Newtonian transport laws;

— electro-magnetic interactions with a fluid;

— plasmas.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this international standard. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this international standard are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO 10303-1:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles.*

ISO 10303-11:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.*

ISO 10303-5w:2000[1)], *Industrial automation systems and integration — Product data representation and exchange — Part 5w: Integrated resource: Mesh-based topology.*

ISO/IEC 8824-1:1995, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

3 Terms, definitions, abbreviations, and symbols

3.1 Terms defined in ISO 10303-1

— application protocol (AP)

— integrated resource (IR)

3.2 Other definitions

3.2.1
BC patch
the subrange of a face of a zone where a given boundary-condition is applied

3.2.2
computational fluid dynamics
the set of knowledge and tools used to generate exact or approximate solutions to the mathematical equations governing the motion of a fluid (gas or liquid).

NOTE 1 The underlying knowledge is implemented in computing codes or application programs.

[1)] To be published.

Table 1 – Symbols for dimensional units

Symbol	Description
M	mass unit
L	length unit
T	time unit
Θ	temperature unit
α	angle unit

Table 2 – Symbols for coordinate systems

Symbol	Description
x, y, z	coordinates in a Cartesian system
r, θ, z	coordinates in a Cylindrical system
r, θ, ϕ	coordinates in a Spherical system
ξ, η, ζ	coordinates in an auxiliary system

3.2.3
global BC data
boundary-condition data applied globally to a BC patch; for example, specifying a uniform total pressure at an inflow boundary

3.2.4
local BC data
boundary-condition data applied at each grid point of a BC patch; an example of this is varying total pressure specified at each vertex of a BC patch

3.3 Abbreviations

CFD computational fluid dynamics

3.4 Symbols

Symbols for dimensional units are given in Table 1.

EXAMPLE 1 A length has dimensions **L**, an area has dimensions $\mathbf{L}^2$, and a velocity has dimensions $\mathbf{L}/\mathbf{T}$ (alternatively written as $\mathbf{L}\mathbf{T}^{-1}$).

Symbols for coordinate systems are given in Table 2.

Associated with the coordinate systems are unit vectors, the symbols for which are given in Table 3.

Symbols for physical properties are given in Table 4.

Table 3 – Symbols for unit vectors

Symbol	Direction	Symbol	Direction	Symbol	Direction
$\hat{e}_x$	x-direction	$\hat{e}_r$	r-direction	$\hat{e}_\xi$	ξ-direction
$\hat{e}_y$	y-direction	$\hat{e}_\theta$	θ-direction	$\hat{e}_\eta$	η-direction
$\hat{e}_z$	z-direction	$\hat{e}_\phi$	ϕ-direction	$\hat{e}_\zeta$	ζ-direction

Table 4 – Symbols for physical properties

Symbol	Description
ρ	static density
p	static pressure
T	static temperature
e	static internal energy per unit mass
h	static enthalpy per unit mass
s	entropy
ρ_0	stagnation density
p_0	stagnation pressure
T_0	stagnation temperature
e_0	stagnation energy per unit mass
h_0	stagnation enthalpy per unit mass
ρe_0	stagnation energy per unit volume
ν	kinematic viscosity ($\nu = \mu/\rho$)
μ	molecular viscosity
ν_t	eddy viscosity
k	thermal conductivity coefficient
R	ideal gas constant ($R = c_p - c_v$)
c_p	specific heat at constant pressure
c_v	specific heat at constant volume

Symbols for nondimensional parameters are given in Table 5

Table 5 – Symbols for nondimensional parameters and related scales

Symbol	Description
M	Mach number ($M = q/c$)
q	Mach velocity scale
c	Mach speed of sound scale
R_e	Reynolds number ($R_e = VL/\nu$)
V	Reynolds velocity scale
L	Reynolds length scale
ν	Reynolds kinematic viscosity scale
Pr	Prandtl number ($Pr = \mu c_p/k$)
k	Prandtl thermal conductivity scale
μ	Prandtl molecular viscosity scale
c_p	Prandtl specific heat scale
γ	specific heat ratio ($\gamma = c_p/c_v$)
c_p	specific heat at constant pressure
c_v	specific heat at constant volume

4 Information requirements

This clause specifies the information required for (TBD — APPLICATION PURPOSE).

The information requirements are specified as a set of units of functionality, application objects, and application assertions. These assertions pertain to individual application objects and to relationships between application objects. The information requirements are defined using the terminology of the subject area of this application protocol.

NOTE 1 A graphical representation of the information requirements is given in annex G.

NOTE 2 The information requirements correspond to those of the activities identified as being within the scope of this application protocol in annex F.

NOTE 3 The mapping table specified in 5.1 shows how the integrated resources are used to meet the information requirements of this application protocol.

4.1 Units of functionality

This subclause specifies the units of functionality for the Fluid dynamics data application protocol. This part of ISO 10303 specifies the following units of functionality:

— UoF1

The units of functionality and a description of the functions that each UoF supports are given below. The application objects included in the UoFs are defined in 4.2.

4.1.1 UoF1

4.2 Application objects

This subclause specifies the application objects for the Fluid dynamics data application protocol. Each application object is an atomic element that embodies a unique application concept and contains attributes specifying the data elements of the object. The application objects and their definitions are given below.

4.3 Application assertions

This subclause specifies the application assertions for the Fluid dynamics data application protocol. Application assertions specify the relationships between application objects, the cardinality of the relationships, and the rules required for the integrity and validity of the application objects and UoFs. The application assertions and their definitions are given below.

5 Application interpreted model

5.1 Mapping table

This clause contains the mapping table that shows how each UoF and application object of this part of ISO 10303 (see clause 4) maps to one or more AIM constructs (see annex A). The mapping table is organized in five columns.

Column 1) Application element: Name of an application element as it appears in the application object definition in 4.2. Application object names are written in uppercase. Attribute names and assertions are listed after the application object to which they belong and are written in lower case.

Column 2) AIM element: Name of an AIM element as it appears in the AIM (see annex A), the term 'IDENTICAL MAPPING', or the term 'PATH'. AIM entities are written in lower case. Attribute names of AIM entities are referred to as <entity name> . <attribute name>. The mapping of an application element may result in several related AIM elements. Each of these AIM elements requires a line of its own in the table. The term 'IDENTICAL MAPPING' indicates that both application objects of an application assertion map to the same AIM element. The term 'PATH' indicates that the application assertion maps to the entire reference path.

Column 3) Source: For those AIM elements that are interpreted from the integrated resources or the application interpreted constructs, this is the number of the corresponding part of ISO 10303. For those AIM elements that are created for the purpose of this part of ISO 10303, this is the number of this part. Entities or types that are defined within the integrated resources have an AIC as the source reference if the use of the entity or type for the mapping is within the scope of the AIC.

Column 4) Rules: One or more numbers may be given that refer to rules that apply to the current AIM element or reference path. For rules that are derived from relationships between application objects, the same rule is referred to by the mapping entries of all the involved AIM elements. The expanded names of the rules are listed after the table.

Column 5) Reference path: To describe fully the mapping of an application object, it may be necessary to specify a reference path through several related AIM elements. The reference path column documents the role of an AIM element relative to the AIM element in the row succeeding it. Two or more such related AIM elements define the interpretation of the integrated resources that satisfies the requirement specified by the application object. For each AIM element that has been created for use within this part of ISO 10303, a reference path up to its supertype from an integrated resource is specified.

For the expression of reference paths the following notational conventions apply:

a) `[]` : enclosed section constrains multiple AIM elements or sections of the reference path are required to satisfy an information requirement;

b) `()` : enclosed section constrains multiple AIM elements or sections of the reference path are identified as alternatives within the mapping to satisfy an information requirement;

c) `{}` : enclosed section constrains the reference path to satisfy an information requirement;

d) `<>` : enclosed section constrains at one or more required reference path;

e) `||` : enclosed section constrains the supertype entity;

f) `->` : attribute references the entity or select type given in the following row;

g) `<-` : entity or select type is referenced by the attribute in the following row;

h) `[i]` : attribute is an aggregation of which a single member is given in the following row;

i) `[n]` : attribute is an aggregation of which member `n` is given in the following row;

j) `=>` : entity is a supertype of the entity given in the following row;

k) `<=` : entity is a subtype of the entity given in the following row;

l) `=` : the string, select, or enumeration type is constrained to a choice or value;

m) `\` : the reference path expression continues on the next line.

5.2 AIM EXPRESS short listing

This clause specifies the EXPRESS schema that uses elements from the integrated resources and contains the types, entity specializations, rules, and functions that are specific to this part of ISO 10303. This clause also specifies modifications to the text for constructs that are imported from the integrated resources. The definitions and EXPRESS provided in the integrated resources for constructs used in the AIM may include select list items and subtypes that are not imported into the AIM. Requirements stated in the integrated resources that refer to select list items and subtypes apply exclusively to those items that are imported into the AIM.

EXPRESS specification:

```
*)
{iso standard 10303 part (11) version (4)}
SCHEMA cfd_aim;
  USE FROM application_context_schema  -- ISO 10303-41 (reqd for any AP)
    (application_context,
     application_context_element,
     product_definition_context
    );
  USE FROM product_definition_schema  -- ISO 10303-41 (reqd for any AP)
    (product,
     product_definition,
     product_definition_formation
    );
  USE FROM product_property_definition_schema  -- ISO 10303-41 (usually reqd)
    (property_definition,
     product_definition_shape
    );
  USE FROM product_property_representation_schema  -- ISO 10303-41 (usually reqd)
    (property_definition_representation,
     shape_definition_representation,
     shape_representation
    );
  USE FROM representation_schema             -- ISO 10303-43 (usually reqd)
    (representation,
     representation_item
    );
  USE FROM external_reference_schema         -- ISO 10304-41
    (externally_defined_item
    );
  USE FROM mesh_topology_schema  -- ISO 10303-5w
    (cell_shape,
     cell_shape_0D,
     cell_shape_1D,
     cell_shape_2D,
     cell_shape_3D,
     mesh_topology,
```



```
    regular_mesh_topology,      -- approx ARM structured_zone
    rectangular_grid,
    cylindrical_grid,
    pyramidal_grid,
    rind,                       -- = ARM
    irregular_mesh_topology,    -- approx ARM unstructured_zone
    cell,
    mesh_topology_data,
    mesh_cell_data,
    mesh_vertex_data
    );
  USE FROM data_array_schema     -- ISO 10303-5w
    (data_class,             -- = ARM
    index,                   -- = ARM INTEGER
    texts,                   -- = ARM LIST OF STRING
    data_name,               -- = ARM
    adhoc_data_name,         -- = ARM
    standard_data_name,      -- = ARM
    coordinate_data_name,    -- = ARM
    other_data_name,
    data_conversion,         -- = ARM
    dimensional_units,       -- = ARM
    index_list,              -- = ARM
    index_range,             -- = ARM
    data_array               -- = ARM
    );
  USE FROM multiblock_schema     -- ISO 10303-5w
    (grid_location,          -- = ARM
    multiblock,              -- = ARM zone_grid_connectivity
    block_connectivity,      -- = ARM connectivity
    matched_connection,
    mismatched_connection,
    mismatched_region,
    abutting,
    overset,
    overset_hole,
    structured_donor,        -- = ARM
    unstructured_donor       -- = ARM
    );
  USE FROM measure_schema        -- ISO 10303-41
    (dimensional_exponents,  -- = ARM  -- REFD into data_array_schema
    si_unit                           -- REFD into data_array_schema
    );
  USE FROM support_resource_schema  -- ISO 10303-41
    (text                     -- REFD into data_array_schema
    );
  USE FROM mathematical_functions_schema  -- ISO 10303-50
    (explicit_table_function, -- REFD into data_array_schema
    listed_real_data,        -- REFD into multiblock_schema
    nonnegative,             -- REFD into mesh_topology_schema
    positive                 -- REFD into mesh_topology_schema
    );
```



```
  USE FROM topology_schema        -- ISO 10303-42
    (topological_representation_item -- REFD into mesh_topology_schema
    );
(*
```

5.2.1 Fundamental concepts and assumptions

5.2.2 Fluid dynamics data types

5.2.2.1 Fluid dynamics data type definitions

5.2.2.1.1 cfd_standard_data_name

A listing of standardized identifiers for the contents of a **data_array**.

EXPRESS specification:

```
*)
TYPE cfd_standard_data_name = SELECT BASED_ON standard_data_name WITH
  (flow_solution_data_name,
   turbulence_data_name,
   nondimensional_data_name,
   Riemann_1D_data_name,
   force_moment_data_name);
(*
```

5.2.2.1.2 flow_solution_data_name

flow_solution_data_name is an enumeration of standardized flow solution data.

EXPRESS specification:

```
*)
TYPE flow_solution_data_name = ENUMERATION OF
     (potential,
      stream_function,
      density,
      pressure,
      temperature,
      energy_internal,
      enthalpy,
      entropy,
      entropy_approx,
      density_stagnation,
      pressure_stagnation,
      temperature_stagnation,
```



```
      energy_stagnation,
      enthalpy_stagnation,
      energy_stagnation_density,
      velocity_x,
      velocity_y,
      velocity_z,
      velocity_r,
      velocity_theta,
      velocity_phi,
      velocity_magnitude,
      velocity_normal,
      velocity_tangential,
      velocity_sound,
      velocity_sound_stagnation,
      momentum_x,
      momentum_y,
      momentum_z,
      momentum_magnitude,
      energy_kinetic,
      pressure_dynamic,
      vorticity_x,
      vorticity_y,
      vorticity_z,
      vorticity_magnitude,
      skin_friction_x,
      skin_friction_y,
      skin_friction_z,
      skin_friction_magnitude,
      velocity_angle_x,
      velocity_angle_y,
      velocity_angle_z,
      velocity_unit_vector_x,
      velocity_unit_vector_y,
      velocity_unit_vector_z,
      mass_flow,
      viscosity_kinematic,
      viscosity_molecular,
      viscosity_eddy,
      thermal_conductivity,
      ideal_gas_constant,
      specific_heat_pressure,
      specific_heat_volume,
      Reynolds_stress_xx,
      Reynolds_stress_xy,
      Reynolds_stress_xz,
      Reynolds_stress_yy,
      Reynolds_stress_yz,
      Reynolds_stress_zz);
END_TYPE;
(*
```

The meanings of the identifiers are given in Table 6.

Table 6 – Flow solution data name identifiers

Data name identifier	Description	Units
potential	potential: $\nabla\phi = \vec{q}$	$\mathbf{L}^2/\mathbf{T}$
stream_function	stream function (2–D): $\nabla\times\psi = \vec{q}$	$\mathbf{L}^2/\mathbf{T}$
density	static density (ρ)	$\mathbf{M}/\mathbf{L}^3$
pressure	static pressure (p)	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
temperature	static temperature (T)	Θ
energy_internal	static internal energy per unit mass (e)	$\mathbf{L}^2/\mathbf{T}^2$
enthalpy	static enthalpy per unit mass (h)	$\mathbf{L}^2/\mathbf{T}^2$
entropy	entropy (s)	$\mathbf{M}\mathbf{L}^2/(\mathbf{T}^2\Theta)$
entropy_approx	approximate entropy ($\tilde{s} = p/\rho^\gamma$)	$\mathbf{L}^{3\gamma-1}/(\mathbf{M}^{\gamma-1}\mathbf{T}^2)$
density_stagnation	stagnation density (ρ_0)	$\mathbf{M}/\mathbf{L}^3$
pressure_stagnation	stagnation pressure (p_0)	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
temperature_stagnation	stagnation temperature (T_0)	Θ
energy_stagnation	stagnation energy per unit mass (e_0)	$\mathbf{L}^2/\mathbf{T}^2$
enthalpy_stagnation	stagnation enthalpy per unit mass (h_0)	$\mathbf{L}^2/\mathbf{T}^2$
energy_stagnation_density	stagnation energy per unit volume (ρe_0)	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
velocity_x	x-component of velocity ($u = \vec{q}\cdot\hat{e}_x$)	$\mathbf{L}/\mathbf{T}$
velocity_y	y-component of velocity ($v = \vec{q}\cdot\hat{e}_y$)	$\mathbf{L}/\mathbf{T}$
velocity_z	z-component of velocity ($w = \vec{q}\cdot\hat{e}_z$)	$\mathbf{L}/\mathbf{T}$
velocity_r	radial velocity component ($\vec{q}\cdot\hat{e}_r$)	$\mathbf{L}/\mathbf{T}$
velocity_theta	velocity component in θ direction ($\vec{q}\cdot\hat{e}_\theta$)	$\mathbf{L}/\mathbf{T}$
velocity_phi	velocity component in ϕ direction ($\vec{q}\cdot\hat{e}_\phi$)	$\mathbf{L}/\mathbf{T}$
velocity_magnitude	velocity magnitude ($q = \sqrt{\vec{q}\cdot\vec{q}}$)	$\mathbf{L}/\mathbf{T}$
velocity_normal	normal velocity component ($\vec{q}\cdot\hat{n}$)	$\mathbf{L}/\mathbf{T}$
velocity_tangential	tangential velocity component (2–D)	$\mathbf{L}/\mathbf{T}$
velocity_sound	static speed of sound	$\mathbf{L}/\mathbf{T}$
velocity_sound_stagnation	stagnation speed of sound	$\mathbf{L}/\mathbf{T}$
momentum_x	x-component of momentum (ρu)	$\mathbf{M}/(\mathbf{L}^2\mathbf{T})$
momentum_y	y-component of momentum (ρv)	$\mathbf{M}/(\mathbf{L}^2\mathbf{T})$
momentum_z	z-component of momentum (ρw)	$\mathbf{M}/(\mathbf{L}^2\mathbf{T})$
momentum_magnitude	magnitude of momentum (ρq)	$\mathbf{M}/(\mathbf{L}^2\mathbf{T})$

Continued on next page

Table 6 — concluded from previous page

Data name identifier	Description	Units
energy_kinetic	$\frac{1}{2}(u^2+v^2+w^2)=\frac{1}{2}q^2$	$\mathbf{L}^2/\mathbf{T}^2$
pressure_dynamic	$\frac{1}{2}\rho q^2$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
vorticity_x	$\omega_x=\partial w/\partial y-\partial v/\partial z=\vec{\omega}\cdot\hat{e}_x$	$\mathbf{T}^{-1}$
vorticity_y	$\omega_y=\partial u/\partial z-\partial w/\partial x=\vec{\omega}\cdot\hat{e}_y$	$\mathbf{T}^{-1}$
vorticity_z	$\omega_z=\partial v/\partial x-\partial u/\partial y=\vec{\omega}\cdot\hat{e}_z$	$\mathbf{T}^{-1}$
vorticity_magnitude	$\omega=\sqrt{\vec{\omega}\cdot\vec{\omega}}$	$\mathbf{T}^{-1}$
skin_friction_x	x-component of skin friction $(\vec{\tau}\cdot\hat{e}_x)$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
skin_friction_y	y-component of skin friction $(\vec{\tau}\cdot\hat{e}_y)$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
skin_friction_z	z-component of skin friction $(\vec{\tau}\cdot\hat{e}_z)$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
skin_friction_magnitude	skin friction magnitude $(\sqrt{\vec{\tau}\cdot\vec{\tau}})$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
velocity_angle_x	velocity angle $(\arccos(u/q)\in[0,\,180°))$	α
velocity_angle_y	$\arccos(v/q)$	α
velocity_angle_z	$\arccos(w/q)$	α
velocity_unit_vector_x	x-component of velocity unit vector $((\vec{q}\cdot\hat{e}_x)/q)$	-
velocity_unit_vector_y	y-component of velocity unit vector $((\vec{q}\cdot\hat{e}_y)/q)$	-
velocity_unit_vector_z	z-component of velocity unit vector $((\vec{q}\cdot\hat{e}_z)/q)$	-
mass_flow	mass flow normal to a plane $(\rho\vec{q}\cdot\hat{n})$	$\mathbf{M}/(\mathbf{L}^2\mathbf{T})$
viscosity_kinematic	kinematic viscosity $(\nu=\mu/\rho)$	$\mathbf{L}^2/\mathbf{T}$
viscosity_molecular	molecular viscosity (μ)	$\mathbf{M}/(\mathbf{L}\mathbf{T})$
viscosity_eddy	eddy viscosity (ν_t)	$\mathbf{L}^2/\mathbf{T}$
thermal_conductivity	thermal conductivity coefficient (k)	$\mathbf{M}\mathbf{L}/(\mathbf{T}^3\Theta)$
ideal_gas_constant	ideal gas constant $(R=c_p-c_v)$	$\mathbf{L}/(\mathbf{T}^2\Theta)$
specific_heat_pressure	specific heat at constant pressure (c_p)	$\mathbf{L}^2/(\mathbf{T}^2\Theta)$
specific_heat_volume	specific heat at constant volume (c_v)	$\mathbf{L}^2/(\mathbf{T}^2\Theta)$
Reynolds_stress_xx	Reynolds stress $-\rho\overline{u'u'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
Reynolds_stress_xy	Reynolds stress $-\rho\overline{u'v'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
Reynolds_stress_xz	Reynolds stress $-\rho\overline{u'w'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
Reynolds_stress_yy	Reynolds stress $-\rho\overline{v'v'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
Reynolds_stress_yz	Reynolds stress $-\rho\overline{v'w'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$
Reynolds_stress_zz	Reynolds stress $-\rho\overline{w'w'}$	$\mathbf{M}/(\mathbf{L}\mathbf{T}^2)$

5.2.2.1.3 turbulence_data_name

turbulence_data_name is an enumeration of standardized Reynolds-averaged Navier-Stokes turbulence model variables.

EXPRESS specification:

```
*)
TYPE turbulence_data_name = ENUMERATION OF
     (turbulent_distance,
      turbulent_energy_kinetic,
      turbulent_dissipation,
```


Table 7 – Turbulence data name identifiers

Data name identifier	Description	Units
turbulent_distance	distance to nearest wall	**L**
turbulent_energy_kinetic	$k = \frac{1}{2}(\overline{u'u'} + \overline{v'v'} + \overline{w'w'})$	$\mathbf{L}^2/\mathbf{T}^2$
turbulent_dissipation	ϵ	$\mathbf{L}^2/\mathbf{T}^3$
turbulent_dissipation_rate	ϵ/k	$\mathbf{T}^{-1}$
turbulent_BB_Reynolds	Baldwin-Barth one-equation model R_T	-
turbulent_SA_nu_tilde	Spalart-Allmaras one-equation model $\tilde{\nu}$	$\mathbf{L}^2/\mathbf{T}$
turbulent_SA_chi	S-A model $\chi = \tilde{\nu}/\nu$	-
turbulent_SA_cb1	S-A model $c_{b1} = 0.1355$	-
turbulent_SA_cb2	S-A model $c_{b2} = 0.622$	-
turbulent_SA_sigma	S-A model $\sigma = 2/3$	-
turbulent_SA_kappa	S-A model $\kappa = 0.41$ (von Karman constant)	-
turbulent_SA_cw1	S-A model $c_{w1} = 3.2391$	-
turbulent_SA_cw2	S-A model $c_{w2} = 0.3$	-
turbulent_SA_cw3	S-A model $c_{w3} = 2$	-
turbulent_SA_cv1	S-A model $c_{v1} = 7.1$	-
turbulent_SA_ct1	S-A model $c_{t1} = 1$	-
turbulent_SA_ct2	S-A model $c_{t2} = 2$	-
turbulent_SA_ct3	S-A model $c_{t3} = 1.2$	-
turbulent_SA_ct4	S-A model $c_{t4} = 0.5$	-

```
      turbulent_dissipation_rate,
      turbulent_BB_Reynolds,
      turbulent_SA_nu_tilde,
      turbulent_SA_chi,
      turbulent_SA_cb1,
      turbulent_SA_cb2,
      turbulent_SA_sigma,
      turbulent_SA_kappa,
      turbulent_SA_cw1,
      turbulent_SA_cw2,
      turbulent_SA_cw3,
      turbulent_SA_cv1,
      turbulent_SA_ct1,
      turbulent_SA_ct2,
      turbulent_SA_ct3,
      turbulent_SA_ct4);
END_TYPE;
(*
```

The meaning of the identifiers is given in Table 7.

5.2.2.1.4 nondimensional_data_name

nondimensional_data_name is an enumeration of standardized nondimensional parameters.

EXPRESS specification:

```
*)
TYPE nondimensional_data_name = ENUMERATION OF
     (Mach,
      Mach_velocity,
      Mach_velocity_sound,
      Reynolds,
      Reynolds_velocity,
      Reynolds_length,
      Reynolds_viscosity_kinematic,
      Prandtl,
      Prandtl_thermal_conductivity,
      Prandtl_viscosity_molecular,
      Prandtl_specific_heat_pressure,
      specific_heat_ratio,
      specific_heat_ratio_pressure,
      specific_heat_ratio_volume,
      coef_pressure,
      coef_skin_friction_x,
      coef_skin_friction_y,
      coef_skin_friction_z,
      coef_pressure_dynamic,
      coef_pressure_reference);
END_TYPE;
(*
```

The meanings of the identifiers are given in Table 8.

5.2.2.1.5 Riemann_1D_data_name

Riemann_1D_data_name is an enumeration of standardized Riemann data for 1-D flow.

EXPRESS specification:

```
*)
TYPE Riemann_1D_data_name = ENUMERATION OF
     (Riemann_invariant_plus,
      Riemann_invariant_minus,
      characteristic_entropy,
      characteristic_vorticity1,
      characteristic_vorticity2,
      characteristic_acoustic_plus,
      characteristic_acoustic_minus);
END_TYPE;
(*
```

The meanings of the enumerated items are given in Table 9.

Table 8 – Nondimensional data name identifiers

Data name identifier	Description	Units
Mach	Mach number: $M = q/c$	-
Mach_velocity	velocity scale (q)	$\mathbf{L/T}$
Mach_velocity_sound	speed of sound scale (c)	$\mathbf{L/T}$
Reynolds	Reynolds number: $Re = VL/\nu$	-
Reynolds_velocity	velocity scale (V)	$\mathbf{L/T}$
Reynolds_length	length scale (L)	$\mathbf{L}$
Reynolds_viscosity_kinematic	kinematic viscosity scale (ν)	$\mathbf{L^2/T}$
Prandtl	Prandtl number: $Pr = \mu c_p/k$	-
Prandtl_thermal_conductivity	thermal conductivity scale (k)	$\mathbf{ML}/(\mathbf{T}^3\Theta)$
Prandtl_viscosity_molecular	molecular viscosity scale (μ)	$\mathbf{M}/(\mathbf{LT})$
Prandtl_specific_heat_pressure	specific heat scale (c_p)	$\mathbf{L}^2/(\mathbf{T}^2\Theta)$
specific_heat_ratio	specific heat ratio: $\gamma = c_p/c_v$	-
specific_heat_ratio_pressure	specific heat at constant pressure (c_p)	$\mathbf{L}^2/(\mathbf{T}^2\Theta)$
specific_heat_ratio_volume	specific heat at constant volume (c_v)	$\mathbf{L}^2/(\mathbf{T}^2\Theta)$
coef_pressure	c_p	-
coef_skin_friction_x	$\vec{c}_f \cdot \hat{e}_x$	-
coef_skin_friction_y	$\vec{c}_f \cdot \hat{e}_y$	-
coef_skin_friction_z	$\vec{c}_f \cdot \hat{e}_z$	-
coef_pressure_dynamic	$1/2\rho_{\text{ref}} q^2_{\text{ref}}$	$\mathbf{M}/(\mathbf{LT}^2)$
coef_pressure_reference	p_{ref}	$\mathbf{M}/(\mathbf{LT}^2)$

Table 9 – Riemann 1-D data name identifiers

Data name identifier	Description	Units
Riemann_invariant_plus	$u + 2c/(\gamma - 1)$	$\mathbf{L/T}$
Riemann_invariant_minus	$u - 2c/(\gamma - 1)$	$\mathbf{L/T}$
characteristic_entropy	$p' - \rho'/\bar{c}^2$	$\mathbf{M}/(\mathbf{LT}^2)$
characteristic_vorticity1	v'	$\mathbf{L/T}$
characteristic_vorticity2	w'	$\mathbf{L/T}$
characteristic_acoustic_plus	$p' + u'/(\bar{\rho}\bar{c})$	$\mathbf{M}/(\mathbf{LT}^2)$
characteristic_acoustic_minus	$p' - u'/(\bar{\rho}\bar{c})$	$\mathbf{M}/(\mathbf{LT}^2)$

5.2.2.1.6 force_moment_data_name

force_moment_data_name is an enumeration of standardized force and moment data.

EXPRESS specification:

```
*)
TYPE force_moment_data_name = ENUMERATION OF
     (force_x,
      force_y,
      force_z,
      force_r,
      force_theta,
      force_phi,
      lift,
      drag,
      moment_x,
      moment_y,
      moment_z,
      moment_r,
      moment_theta,
      moment_phi,
      moment_xi,
      moment_eta,
      moment_zeta,
      moment_center_x,
      moment_center_y,
      moment_center_z,
      coef_lift,
      coef_drag,
      coef_moment_x,
      coef_moment_y,
      coef_moment_z,
      coef_moment_r,
      coef_moment_theta,
      coef_moment_phi,
      coef_moment_xi,
      coef_moment_eta,
      coef_moment_zeta,
      coef_moment_pressure_dynamic,
      coef_moment_area,
      coef_length);
END_TYPE;
(*
```

The meanings of the enumerated items are given in Table 10.

Table 10 – Force and moment data name identifiers

Data name identifier	Description	Units
force_x	$F_x = \vec{F} \cdot \hat{e}_x$	$\mathbf{ML/T^2}$
force_y	$F_y = \vec{F} \cdot \hat{e}_y$	$\mathbf{ML/T^2}$
force_z	$F_z = \vec{F} \cdot \hat{e}_z$	$\mathbf{ML/T^2}$
force_r	$F_r = \vec{F} \cdot \hat{e}_r$	$\mathbf{ML/T^2}$
force_theta	$F_\theta = \vec{F} \cdot \hat{e}_\theta$	$\mathbf{ML/T^2}$
force_phi	$F_\phi = \vec{F} \cdot \hat{e}_\phi$	$\mathbf{ML/T^2}$
lift	L or L'	$\mathbf{ML/T^2}$
drag	D or D'	$\mathbf{ML/T^2}$
moment_x	$M_x = \vec{M} \cdot \hat{e}_x$	$\mathbf{ML^2/T^2}$
moment_y	$M_y = \vec{M} \cdot \hat{e}_y$	$\mathbf{ML^2/T^2}$
moment_z	$M_z = \vec{M} \cdot \hat{e}_z$	$\mathbf{ML^2/T^2}$
moment_r	$M_r = \vec{M} \cdot \hat{e}_r$	$\mathbf{ML^2/T^2}$
moment_theta	$M_\theta = \vec{M} \cdot \hat{e}_\theta$	$\mathbf{ML^2/T^2}$
moment_phi	$M_\phi = \vec{M} \cdot \hat{e}_\phi$	$\mathbf{ML^2/T^2}$
moment_xi	$M_\xi = \vec{M} \cdot \hat{e}_\xi$	$\mathbf{ML^2/T^2}$
moment_eta	$M_\eta = \vec{M} \cdot \hat{e}_\eta$	$\mathbf{ML^2/T^2}$
moment_zeta	$M_\zeta = \vec{M} \cdot \hat{e}_\zeta$	$\mathbf{ML^2/T^2}$
moment_center_x	$x_0 = \vec{r}_0 \cdot \hat{e}_x$	**L**
moment_center_y	$y_0 = \vec{r}_0 \cdot \hat{e}_y$	**L**
moment_center_z	$z_0 = \vec{r}_0 \cdot \hat{e}_z$	**L**
coef_lift	C_L or c_l	-
coef_drag	C_D or c_d	-
coef_moment_x	$\vec{C}_M \cdot \hat{e}_x$ or $\vec{c}_m \cdot \hat{e}_x$	-
coef_moment_y	$\vec{C}_M \cdot \hat{e}_y$ or $\vec{c}_m \cdot \hat{e}_y$	-
coef_moment_z	$\vec{C}_M \cdot \hat{e}_z$ or $\vec{c}_m \cdot \hat{e}_z$	-
coef_moment_r	$\vec{C}_M \cdot \hat{e}_r$ or $\vec{c}_m \cdot \hat{e}_r$	-
coef_moment_theta	$\vec{C}_M \cdot \hat{e}_\theta$ or $\vec{c}_m \cdot \hat{e}_\theta$	-
coef_moment_phi	$\vec{C}_M \cdot \hat{e}_\phi$ or $\vec{c}_m \cdot \hat{e}_\phi$	-
coef_moment_xi	$\vec{C}_M \cdot \hat{e}_\xi$ or $\vec{c}_m \cdot \hat{e}_\xi$	-
coef_moment_eta	$\vec{C}_M \cdot \hat{e}_\eta$ or $\vec{c}_m \cdot \hat{e}_\eta$	-
coef_moment_zeta	$\vec{C}_M \cdot \hat{e}_\zeta$ or $\vec{c}_m \cdot \hat{e}_\zeta$	-
coef_pressure_dynamic	$1/2 \rho_{\text{ref}} q^2_{\text{ref}}$	$\mathbf{M/(LT^2)}$
coef_area	S_{ref}	$\mathbf{L^2}$
coef_length	c_{ref}	**L**

5.2.2.1.7 bc_type

Boundary-condition types identify the equations that should be enforced at a given boundary location. The boundary-condition types are described by **bc_type**. Some members of **bc_type** completely identify the equations to impose, while others identify a general description of the class of boundary-condition equations to impose.

bc_type is subdivided into two enumeration types: **bc_type_simple** and **bc_type_compound** which identify the simple and compound boundary-condition types respectively.

The subdivision of **bc_type** is based on function. For simple boundary-conditions, the equations and data are fixed; whereas, for compound boundary-conditions different sets of equations are imposed depending on local flow conditions at the boundary.

bc_type is a superset of **bc_type_simple** and **bc_type_compound**. It identifies the boundary-condition (simple or compound) at a boundary location.

For inflow/outflow boundary-condition descriptions, 3-D inviscid compressible flow is assumed; the 2-D equivalent should be obvious. These same boundary-conditions are typically used for viscous cases also. This '3-D Euler' assumption will be noted wherever used.

EXPRESS specification:

```
*)
TYPE bc_type = SELECT
     (bc_type_simple,
      bc_type_compound);
END_TYPE;
(*
```

5.2.2.1.8 bc_type_simple

bc_type_simple is an enumeration type that identifies the simple boundary-condition at a boundary location.

In the descriptions below, Q is the solution vector, $\vec{q}$ is the velocity vector whose magnitude is q, the unit normal to the boundary is $\hat{n}$, and $\partial()/\partial n = \hat{n} \cdot \nabla$ is differentiation normal to the boundary.

EXPRESS specification:

```
*)
TYPE bc_type_simple = ENUMERATION OF
     (bc_general,
```



```
      bc_Dirichlet,
      bc_Neumann,
      bc_extrapolate,
      bc_wall_inviscid,
      bc_wall_viscous_heat_flux,
      bc_wall_viscous_isothermal,
      bc_wall_viscous,
      bc_wall,
      bc_inflow_subsonic,
      bc_inflow_supersonic,
      bc_outflow_subsonic,
      bc_outflow_supersonic,
      bc_tunnel_inflow,
      bc_tunnel_outflow,
      bc_degenerate_line,
      bc_degenerate_point,
      bc_symmetry_plane,
      bc_symmetry_polar,
      bc_axissymmetric_wedge);
END_TYPE;
(*
```

Enumerated item definitions:

bc_general: arbitrary conditions on Q or $\partial Q/\partial n$;

bc_Dirichlet: Dirichlet condition on Q vector;

bc_Neumann: Neumann condition on $\partial Q/\partial n$;

bc_extrapolate: extrapolate Q from interior;

bc_wall_inviscid: inviscid (slip) wall

— normal velocity specified (default: $\vec{q} \cdot \hat{n} = 0$)

bc_wall_viscous_heat_flux: viscous no-slip wall with heat flux

— velocity Dirichlet (default: $q = 0$)

— temperature Neumann (default: adiabatic, $\partial T/\partial n = 0$)

bc_wall_viscous_isothermal: viscous no-slip, isothermal wall

— velocity Dirichlet (default: $q = 0$)

— temperature Dirichlet

bc_wall_viscous: viscous no-slip wall; special cases are **bc_wall_viscous_heat_flux** and **bc_wall_viscous_isothermal**.

— velocity Dirichlet (default: $q = 0$)

— Dirichlet or Neumann on temperature

bc_wall: general wall condition; special cases are **bc_wall_inviscid**, **bc_wall_viscous**, **bc_wall_viscous_heat_flux**, and **bc_wall_viscous_isothermal**

bc_inflow_subsonic: inflow with subsonic normal velocity

— specify 4; extrapolate 1 (3-D Euler)

bc_inflow_supersonic: inflow with supersonic normal velocity

— specify 5; extrapolate 0 (3-D Euler)

same as **bc_Dirichlet**

bc_outflow_subsonic: outflow with subsonic normal velocity

— specify 1; extrapolate 0 (3-D Euler)

bc_outflow_supersonic: outflow with supersonic normal velocity

— specify 0; extrapolate 5 (3-D Euler)

same as **bc_Extrapolate**

bc_tunnel_inflow: tunnel inlet (subsonic normal velocity)

— specify cross-flow velocity, stagnation enthalpy, entropy

— extrapolate 1 (3-D Euler)

bc_tunnel_outflow: tunnel exit (subsonic normal velocity)

— specify static pressure

— extrapolate 4 (3-D Euler)

bc_degenerate_line: face degenerated to a line;

bc_degenerate_point: face degenerated to a point;

bc_symmetry_Plane: symmetry plane; face should be coplanar

— density, pressure: $\partial()/\partial n = 0$

— tangential velocity: $\partial(\vec{q} \times \hat{n})/\partial n = 0$

— normal velocity: $\vec{q} \cdot \hat{n} = 0$

bc_symmetry_polar: polar-coordinate singularity line; special case of **bc_degenerate_line** where degenerate face is a straight line and flowfield has polar symmetry; $\hat{s}$ is singularity line tangential unit vector

— normal velocity: $\vec{q} \times \hat{s} = 0$

— all others: $\partial()/\partial n = 0$, n normal to $\hat{s}$

bc_axissymmetric_wedge: axisymmetric wedge; special case of **bc_degenerate_line** where degenerate face is a straight line

5.2.2.1.9 bc_type_compound

bc_type_compound is an enumeration type that identifies the compound boundary-condition at a boundary location.

EXPRESS specification:

```
*)
TYPE bc_type_compound = ENUMERATION OF
     (bc_inflow,
      bc_outflow,
      bc_farfield);
END_TYPE;
(*
```

Enumerated item definitions:

bc_inflow: inflow, arbitrary normal Mach
test on normal Mach, then perform one of: **bc_inflow_subsonic**, **bc_inflow_supersonic**;

bc_outflow: outflow, arbitrary normal Mach
test on normal Mach, then perform one of: **bc_outflow_subsonic**, **bc_outflow_supersonic**;

bc_farfield: farfield inflow/outflow, arbitrary normal Mach
test on normal velocity and normal Mach, then perform one of: **bc_inflow_subsonic**, **bc_inflow_supersonic**, **bc_outflow_subsonic**, **bc_outflow_supersonic**.

5.2.2.1.10 governing_equations_type

governing_equations_type is an enumeration of the classes of flow equations.

EXPRESS specification:

```
*)
TYPE governing_equations_type = ENUMERATION OF
     (unspecified,
      full_potential,
      Euler,
      NS_laminar,
      NS_turbulent,
```



```
      NS_laminar_incompressible,
      NS_turbulent_incompressible);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

full_potential: is full potential flow;

Euler: is Euler flow;

NS_laminar: is Navier-Stokes laminar flow;

NS_turbulent: is Navier-Stokes turbulent flow;

NS_laminar_incompressible: is Navier-Stokes laminar incrompressible flow;

NS_turbulent_incompressible: is Navier-Stokes turbulent incrompressible flow;

5.2.2.1.11 gas_model_type

gas_model_type is an enumeration of the state models relating pressure, temperature and density.

EXPRESS specification:

```
*)
TYPE gas_model_type = ENUMERATION OF
     (unspecified,
      ideal,
      Van_der_Waals);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified.

ideal: the state model is the perfect gas law. The pressure, temperature and density are related by,

$$p = \rho RT,$$

where R is the ideal gas constant. Related quantities are the specific heat at constant pressure (c_p), specific heat at constant volume (c_v) and specific heat ratio ($\gamma = c_p/c_v$). The gas constant and specific heats are related by $R = c_p - c_v$.

The **standard_data_name** identifiers associated with the perfect gas law are: **ideal_gas_constant**, **specific_heat_ratio**, **specific_heat_volume** and **specific_heat_pressure**. These are described in clause G.3.3.12.

Van_der_Waals: the state model is Van der Waals' equation.

5.2.2.1.12 viscosity_model_type

viscosity_model_type is an enumeration of the relationships between molecular viscosity and temperature.

EXPRESS specification:

```
*)
TYPE viscosity_model_type = ENUMERATION OF
     (unspecified,
      constant_viscosity,
      power_law,
      Sutherland_law);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

constant_viscosity: the molecular viscosity is constant throughout the field and is equal to some reference value ($\mu = \mu_{\text{ref}}$)

power_law: the molecular viscosity follows a power-law relation,

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^n .$$

The **standard_data_name** identifiers associated with this model are: **power_law_exponent**, **temperature_reference** and **viscosity_molecular_reference**. These are described in clause G.3.3.12.

Sutherland_law: Sutherland's Law for molecular viscosity,

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{3/2} \frac{T_{\text{ref}} + T_s}{T + T_s},$$

where T_s is the Sutherland Law constant, and μ_{ref} and T_{ref} are the reference viscosity and temperature, respectively.

The **standard_data_name** identifiers associated with this model are: **Sutherland_law_constant**, **temperature_reference** and **viscosity_molecular_reference**. These are described in clause G.3.3.12.

NOTE 1

For air [4], the power-law exponent is $n = 0.666$, Sutherlands Law constant (T_s) is 110.6 K, the reference temperature (T_{ref}) is 273.15 K, and the reference viscosity (μ_{ref}) is 1.716×10^{-5} kg/(m s).

5.2.2.1.13 thermal_conductivity_model_type

thermal_conductivity_model_type is an enumeration of the relationships between the thermal-conductivity coefficient and temperature.

EXPRESS specification:

```
*)
TYPE thermal_conductivity_model_type = ENUMERATION OF
     (unspecified,
      constant_Prandtl,
      power_law,
      Sutherland_law);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

constant_Prandtl: the Prandtl number ($Pr = \mu c_p/k$) is constant and equal to some reference value.

The **standard_data_name** identifier associated with this model is **constant_Prandtl**, and is described in clause G.3.3.12.

power_law: the thermal conductivity is related to temperature via a power-law.

$$k = k_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^n .$$

The **standard_data_name** identifiers associated with this model are: **power_law_exponent**, **temperature_reference** and **thermal_conductivity_reference**. These are described in clause G.3.3.12.

Sutherland_law: Sutherland's Law for thermal conductivity.

$$k = k_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{3/2} \frac{T_{\text{ref}} + T_s}{T + T_s},$$

where T_s is the Sutherland Law constant, and k_{ref} and T_{ref} are the reference thermal conductivity and temperature, respectively.

The **standard_data_name** identifiers associated with this model are: **Sutherland_law_constant**, **temperature_reference** and **thermal_conductivity_molecular_reference**. These are described in clause G.3.3.12.

NOTE 1

For air [4], the Prandtl number is $Pr = 0.72$, the power-law exponent is $n = 0.81$, Sutherlands Law constant (T_s) is 194.4 K, the reference temperature (T_{ref}) is 273.15 K, and the reference thermal conductivity (k_{ref}) is 2.414×10^{-2} $kg\ m/(s^3 K)$.

5.2.2.1.14 turbulence_closure_type

turbulence_closure_type is an enumeration of the kinds of turbulence closure for the Reynolds stress terms of the Navier-Stokes equations.

EXPRESS specification:

```
*)
TYPE turbulence_closure_type = ENUMERATION OF
     (unspecified,
      eddy_viscosity,
      Reynolds_stress,
      Reynolds_stress_algebraic);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

eddy_viscosity: Boussinesq eddy-velocity closure. The Reynolds stresses are approximated as the product of an eddy viscosity (ν_t) and the mean strain tensor. Using indicial notation, the relation is,

$$-\overline{u_i' u_j'} = \nu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

where $-\overline{u_i' u_j'}$ are the Reynolds stresses.

Reynolds_stress: no approximation of the Reynolds stresses.

Reynolds_stress_algebraic: an algebraic approximation for the Reynolds stresses based on some intermediate transport quantities.

The associated **standard_data_name** name identifiers are: **eddy_viscosity** and **Prandtl_-turbulent**. These are described in clause G.3.3.12.

5.2.2.1.15 turbulence_model_type

turbulence_model_type is an enumeration of the equation sets for modeling the turbulence quantities.

EXPRESS specification:

```
*)
TYPE turbulence_model_type = ENUMERATION OF
     (unspecified,
      algebraic_Baldwin_Lomax,
      algebraic_Cebeci_Smith,
      half_equation_Johnson_King,
      one_equation_Baldwin_Barth,
      one_equation_Spalart_Allmaras,
      two_equation_Jones_Launder,
      two_equation_Menter_SST,
      two_equation_Wilcox);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

algebraic_Baldwin_Lomax: is Baldwin-Lomax;

algebraic_Cebeci_Smith: is Cebeci-Smith;

half_equation_Johnson_King: is Johnson-King;

one_equation_Baldwin_Barth: is Baldwin-Barth;

one_equation_Spalart_Allmaras: is Spalart-Allmaras;

two_equation_Jones_Launder: is Jones-Launder;

two_equation_Menter_SST: is Menter;

two_equation_Wilcox: is Wilcox.

The associated **standard_data_name** name identifiers for the Spalart-Allmaras turbulence model (version Ia) are: **turbulent_SA_cb1**, **turbulent_SA_cb2**, **turbulent_SA_sigma**, **turbulent_SA_kappa**, **turbulent_SA_cw1**, **turbulent_SA_cw2**, **turbulent_SA_cw3**, **turbulent_SA_cv1**, **turbulent_SA_ct1**, **turbulent_SA_ct2**, **turbulent_SA_ct3**, and **turbulent_SA_ct4**. These are described in clause G.3.3.12.

5.2.2.2 Fluid dynamics data imported type definitions

5.2.2.2.1 TYPE 1

5.2.2.3 Fluid dynamics data entity definitions

5.2.2.3.1 cfd_base

The highest level structure in a CFD database is **cfd_base**. It contains the dimensionality of the grid and a list of zones making up the domain. Globally applicable information, including a reference state, a set of flow equations, dimensional units, and convergence history are also attached. In addition, structures for describing or annotating the database are also accomodated.

EXPRESS specification:

```
*)
ENTITY cfd_base;
  description        : OPTIONAL texts;  -- ARM LIST OF STRING;
  cell_dimension     : positive;           -- ARM INTEGER;
  physical_dimension : positive;           -- ARM INTEGER;
  zones              : LIST OF zone;
  refstate           : OPTIONAL reference_state;
  class              : OPTIONAL data_class;
  units              : OPTIONAL dimensional_units;
  equations          : OPTIONAL flow_equation_set;
  history            : OPTIONAL convergence_history;
  data               : LIST OF integral_data;
  families           : LIST OF family;
END_ENTITY;
(*
```

Attribute definitions:

description: Annotations;

cell_dimension: The dimension of cells in the mesh.

physical_dimension: The number of coordinates required to define a node position.

zones: Data specific to each zone or block in a multiblock case. The size of the list defines the number of zones or blocks in the domain.

refstate: Reference data applicable to the entire database; quantities such as Reynolds number and freestream Mach number are given here (for external flow problems).

class: Global default data class for the database. If the CFD database contains dimensional data (e.g., velocity with units of m/s), **units** may be used to describe the system of units employed.

units: Specification of the global default units;

equations: Description of the governing flow equations associated with the entire database. This structure contains information on the general class of governing equations (e.g., Euler or Navier-Stokes), equation sets required for closure, including turbulence modelling and equations of state, and constants associated with the equations.

history: Global relevant convergence history. The convergence information includes total configuration forces, global parameters (e.g., freestream angle-of-attack), and global residual and solution-change norms taken over all the zones.

data: Miscellaneous data. Candidates for inclusion are global forces and moments.

families: Global family information.

class, **units**, **refstate** and **equations** have special function in the CFD hierarchy. They are globally applicable throughout the database, but their values may be superseded by local entities (e.g., within a given zone).

5.2.2.3.2 zone

zone contains all information pertinent to an individual multiblock zone. This information includes the number of cells and vertices making up the grid, the physical coordinates of the grid vertices, the flow solution, multiblock interface connectivity, boundary-conditions, and zonal convergence-history data. In addition this structure contains a reference state, a set of flow equations and dimensional units that are all unique to the zone.

EXPRESS specification:

```
*)
ENTITY zone;
  description       : OPTIONAL texts;  -- ARM LIST OF STRING;
  grid              : mesh_topology;
  coordinates       : OPTIONAL grid_coordinates;
  family_name       : OPTIONAL family;
  solution          : LIST OF flow_solution;
  field_data        : LIST OF discrete_data;
  global_data       : LIST OF integral_data;
  grid_connectivity : OPTIONAL multiblock; -- ARM zone_grid_connectivity;
  conditions        : OPTIONAL zone_bc;
  rstate            : OPTIONAL reference_state;
  dclass            : OPTIONAL data_class;
  dimunits          : OPTIONAL dimensional_units;
  floweqset         : OPTIONAL flow_equation_set;
  history           : OPTIONAL convergence_history;
DERIVE
  cell_dimension     : positive := base.cell_dimension;
  physical_dimension : positive := base.physical_dimension;
  dimension          : positive := grid.dimension;
```



```
  vertex_size        : ARRAY [1:dimension] OF positive := grid.vertex_size;
  cell_size          : ARRAY [1:dimension] OF positive := grid.cell_size;
  class              : data_class := NVL(dclass, base.class);
  units              : dimensional_units := NVL(dimunits, base.units);
  equations          : flow_equation_set:= NVL(floweqset, base.equations);
  refstate           : reference_state := NVL(rstate, base.refstate);
INVERSE
  base : cfd_base FOR zones;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_zone FOR zone;
  ABSTRACT SUPERTYPE;
  ONEOF(structured_zone,
        unstructured_zone);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

description: is annotation;

grid: the grid;

coordinates: are the physical coordinates of the grid vertices. This structure defines 'the grid'; it may optionally contain physical coordinates of rind or ghost points.

family_name: Identifies to which family the zone belongs to. Family names may be used to define material properties.

solution: is the flow-solution quantities. Each instance of **flow_solution** shall only contain data at a single grid location (vertices, cell-centers, etc.); therefore, multiple **flow_solution** structures are provided to store flow-solution data at different grid locations. These structures may optionally contain solution data defined at rind points.

field_data: is miscellaneous field data. Candidate information includes residuals, fluxes and other discrete data that is considered auxiliary to the flow solution.

global_data: is miscellaneous zone-specific global data, other than reference-state data and convergence history information.

grid_connectivity: is the multiblock interface-connectivity information.

conditions: is the boundary-condition information.

rstate: non-default reference-state data.

dclass: non-default class of data.

dimunits: non-default system of units.

floweqset: if a set of flow equations are specific to an individual zone, these are described here.

EXAMPLE 1 For example, if a single zone in the domain is inviscid, whereas all others are turbulent, then this zone-specific equation set could be used to describe the special zone.

history: is the convergence history of the zone; this includes residual and solution-change norms.

cell_dimension: The dimension of a cell in the mesh.

physical_dimension: The number of coordinates required to define a node position.

dimension: The number of indices required to identify uniquely a vertex or a cell in the grid. It is the dimensionality of the computational grid. For structured-grid calculations, **dimension** is usually the same as the spatial problem being solved (e.g., **dimension**=3 for a 3-D problem). For lower-dimensional flowfields, such as quasi 3-D flow, **dimension** may not be the same as the dimensionality of the position vector or the velocity vector. For unstructured grids, usually **dimension**=1 since all the grid points and flow solution variables are stored in 1-D arrays. However, there are instances, such as prismatic boundary-layer grids, where **dimension** may be 2.

vertex_size: is the number of vertices in each index direction. it is the number of vertices defining 'the grid' or the domain (i.e., without rind points).

cell_size: is the number of cells in each index direction. It is the number of cells on the interior of the domain.

class: is the zonal default for the class of data contained in the zone and its substructures.

units: is the description of the system of dimensional units in the zone.

refstate: is reference-state data specific to the individual zone.

equations: is the flow equation set.

base: is the database.

5.2.2.3.3 structured_zone

structured_zone contains the information pertinent to an individual structured multiblock zone.

EXPRESS specification:

```
*)
ENTITY structured_zone
  SUBTYPE OF (zone);
  SELF\zone.grid : regular_mesh_topology;
END_ENTITY;
(*
```


Attribute definitions:

grid: the structured grid.

5.2.2.3.4 unstructured_zone

unstructured_zone contains the information pertinent to an individual unstructured zone.

EXPRESS specification:

```
*)
ENTITY unstructured_zone
  SUBTYPE OF (zone);
  SELF\zone.grid : unstructured_mesh_topology;
END_ENTITY;
(*
```

Attribute definitions:

grid: the unstructured grid.

5.2.2.3.5 grid_coordinates

The physical coordinates of the grid vertices in a zone are described by the **grid_coordinates** structure. The structure contains a list for the data arrays of the individual components of the position vector. It also provides a mechanism for identifying rind-point data included within the position-vector arrays.

EXPRESS specification:

```
*)
ENTITY grid_coordinates
  SUBTYPE OF (mesh_topology_data);
  rind        : OPTIONAL rind;
  SELF\mesh_topology_data.data : ARRAY [1:physical_dimension] OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  dimension   : positive := zone.dimension;
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  vertex_size : ARRAY [1:dimension] OF INTEGER := zone.vertex_size;
  data_size   : ARRAY [1:dimension] OF positive := grid_data_size(SELF);
  grid        : mesh_topology := zone.grid;
INVERSE
```



```
  zone : zone FOR coordinates;
WHERE
  wr1 : (NOT EXISTS(rind)) XOR (rind.dimension = dimension);
  wr2 : NOT ((schdot+'UNSTRUCTURED_ZONE' IN TYPEOF(zone)) AND EXISTS(rind));
  wr3 : SELF\mesh_topology_data.mesh :=: grid;
  wr3 : SIZEOF(QUERY(v <* data) |
              consistent_data_array(v,
                                    dimension,
                                    data_size,
                                    coordinate_data_name,
                                    class, units, ?, ?))
       = SIZEOF(data);
END_ENTITY;
(*
```

Attribute definitions:

rind: is optional. If not given then this is equivalent to a **rind** structure whose **planes** array contains all zeros.

data: is the grid-coordinate data; each **data_array** shall contain a single component of the position vector (e.g., three structures are required for 3-D data, one for each coordinate value).

dclass: non-default data class;

dimunits: non-default system of units;

dimension: The number of indices required to reference a node.

class: is the default class for data contained in **data_array**.

units: describes the system of units employed.

vertex_size: is the number of vertices, excluding rind points, in each index direction;

grid: is the grid for which the coordinates are specified;

zone: is the calling zone.

Formal propositions:

wr1: If **rind** has a value, then the value of **rind.dimension** shall be equal to the value of **dimension**.

wr2: Grid coordinates for an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

wr3: The attribute values of **data** shall be consistent; in particular, the **identifier** shall be a **coordinate_data_name**.

5.2.2.3.6 flow_solution

The flow solution within a given zone is described by the **flow_solution** structure. This structure contains a list of the data arrays of the individual flow solution variables, as well as identifying the grid location of the solution. It also provides a mechanism for identifying rind-point data included within the data arrays.

EXPRESS specification:

```
*)
ENTITY flow_solution
  SUBTYPE OF(mesh_topology_data);
  rind        : OPTIONAL rind;
  data        : LIST OF data_array;
  gridloc     : OPTIONAL grid_location;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  dimension   : positive := zone.dimension;
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  vertex_size : ARRAY [1:dimension] OF INTEGER := zone.vertex_size;
  cell_size   : ARRAY [1:dimension] OF INTEGER := zone.cell_size;
  location    : grid_location := NVL(gridloc, vertex);
  grid        : mesh_topology := zone.grid;
  field_size  : ARRAY [1:dimension] OF positive :=
                field_data_size(dimension, vertex_size, cell_size,
                                location, rind);
INVERSE
  zone : zone FOR solution;
WHERE
  wr1 : (NOT EXISTS(rind)) XOR (rind.dimension = dimension);
  wr2 : NOT ((schdot+'UNSTRUCTURED_ZONE' IN TYPEOF(zone)) AND EXISTS(rind));
  wr3 : SELF\mesh_topology_data.mesh :=: grid;
  wr3 : SIZEOF(QUERY(v <* data) |
               consistent_data_array(v,
                                     dimension,
                                     field_size,
                                     ?,
                                     class, units, ?, ?))
        = SIZEOF(data);
  wr4 : (location = vertex) XOR (location = cell_center);
END_ENTITY;
(*
```

Attribute definitions:

rind: is the number of rind planes included in the data.

data: is the data. Each structure in the list contains a single component of the solution vector.

gridloc: is the non-default kind of grid location;

dclass: non-default class of data;

dimunits: non-default system of units;

dimension: The number of indices required to reference a node.

class: is the default class for data in DataArrays.

units: is the description of the system of units.

vertex_size: is the numbers of core vertices in each index direction.

cell_size: is the numbers of core cells in each index direction.

location: specifies the location of the solution data with respect to the grid. All data within a given instance of **flow_solution** resides at the same kind of grid location.

grid: is the grid fow which the flow solution is specified;

field_size: is the size of the flow solution data arrays;

zone: is the zone.

Formal propositions:

wr1: If **rind** has a value, then the value of **rind.dimension** shall be equal to the value of **dimension**.

wr2: A flow solution of an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

wr3: **DataArrays** shall be consistent.

wr4: The grid location shall be either **vertex** or **cell_center**.

5.2.2.3.7 zone_bc

All boundary-condition information pertaining to a given zone is contained in the **zone_bc** structure.

EXPRESS specification:

```
*)
ENTITY zone_bc;
  description : OPTIONAL texts;
  conditions  : LIST OF bc;
  rstate      : OPTIONAL reference_state;
  dclass      : OPTIONAL data_class;
```



```
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  dimension          : positive := zone.dimension;
  physical_dimension : positive := zone.physical_dimension;
  class              : data_class := NVL(dclass, zone.class);
  units              : dimensional_units := NVL(dimunits, zone.units);
  refstate           : reference_state := NVL(rstate, zone.refstate);
  grid               : mesh_topology := zone.grid;
INVERSE
  zone : zone FOR conditions;
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

conditions: is the boundary-conditions for a zone, on a patch by patch basis. Boundary-condition information for a single patch is contained in the **bc** structure. If a zone contains N boundary-condition patches, then N seperate instances of **bc** shall be provided in the **zone_bc** entity for the zone.

rstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

dimension: The number of indices required to reference a node.

physical_dimension: The number of coordinates required to define a node position.

refstate: is reference data applicable to all the boundary-condition of the zone. Reference state data is useful for situations where boundary-condition data is not provided, and flow solvers are free to enforce any appropriate boundary-condition equations.

EXAMPLE 1 An engine nozzle exit boundary-condition usually imposes a stagnation pressure (or some other stagnation quantity) different from freestream. The nozzle-exit stagnation quantities could be specified by **refstate** at this level or below in lieu of providing explicit Dirichlet or Neumman data.

class: is the zonal default for the class of data contained in the zone's boundary-conditions.

units: is the system of dimensional units.

grid: is the grid for which the boundary-condition information is specified;

zone: is the zone.

5.2.2.3.8 bc

bc contains boundary-condition information for a single BC surface patch of a zone. A BC patch is the subrange of the face of a zone where a given boundary-condition is applied.

The structure contains a boundary-condition type, as well as one or more sets of boundary-condition data that are used to define the boundary-condition equations to be enforced on the BC patch. For most boundary-conditions, a single data set is all that is needed. The structure also contains information describing the normal vector to the BC surface patch.

EXPRESS specification:

```
*)
ENTITY bc;
  description             : OPTIONAL texts;
  the_type                : bc_type;
  point_range             : OPTIONAL index_range;
  point_list              : OPTIONAL index_list;
  elements                : OPTIONAL LIST OF elements;
  element_range           : OPTIONAL index_range;
  element_list            : OPTIONAL index_list;
  inward_normal_index     : OPTIONAL ARRAY [1:dimension] OF INTEGER;
  inward_normal_list      : OPTIONAL index_list;
  data_sets               : LIST OF bc_data_set;
  family_name             : OPTIONAL family;
  rstate                  : OPTIONAL reference_state;
  dclass                  : OPTIONAL data_class;
  dimunits                : OPTIONAL dimensional_units;
--  vertex_list_length      : positive;
  face_center_list_length : OPTIONAL positive;
DERIVE
  dimension          : positive := zonebc.dimension;
  physical_dimension : positive := zonebc.physical_dimension;
  refstate           : reference_state := NVL(refstate, zonebc.refstate);
  class              : data_class := NVL(dclass, zonebc.class);
  units              : dimensional_units := NVL(dimunits, zonebc.units);
INVERSE
  zonebc : zone_bc FOR conditions;
WHERE
  wr1 : EXISTS(point_range) XOR EXISTS(point_list) XOR
        EXISTS(elements) XOR EXISTS(element_range) XOR
        EXISTS(element_list);
  wr2 : (NOT EXISTS(point_range)) XOR (point_range.dimension = dimension);
  wr3 : (NOT EXISTS(point_list)) XOR (point_list.dimension = dimension);
--        (IndexArrayOK(PointList, Int, dimension, VertexListLength));
  wr4 : (NOT EXISTS(InwardNormalList)) XOR
        (IndexArrayOK(InwardNormalList, Float, physical_dimension, VertexListLength));
  wr5 : (NOT EXISTS(data_sets)) XOR
        (EXISTS(data_sets) AND EXISTS(face_center_list_length));
END_ENTITY;
(*
```


Table 11 – InwardNormalIndex values

face	**InwardNormalIndex**	face	**InwardNormalIndex**
i-min	$[+1, 0, 0]$	i-max	$[-1, 0, 0]$
j-min	$[0, +1, 0]$	j-max	$[0, -1, 0]$
k-min	$[0, 0, +1]$	k-max	$[0, 0, -1]$

Attribute definitions:

description: is annotations;

the_type: is the type of the boundary-condition;

point_range: is a face subrange (i.e., points in a single computational plane); by convention the indices refer to vertices;

point_list: is a face subrange (i.e., points in a single computational plane); by convention the indices refer to vertices;

elements: is the elements;

element_range: is an element subrange;

element_list: is the element indices;

inward_normal_index: shall have only one non-zero element, whose sign indicates the computational-coordinate direction of the BC patch normal; this normal points into the interior of the zone.

Some boundary-conditions require a normal direction to be specified in order to be properly imposed. A computational-coordinate normal can be derived from **point_range** or **point_list** by examining redundant index components. Alternatively, this information can be provided directly by **inward_normal_index**. For exterior faces of a zone in 3-D, **inward_normal_index** takes one of the values given in Table 11.

inward_normal_list: is a list of vectors normal to the BC patch pointing into the interior of the zone; the vectors are not required to be unit vectors. By convention the vectors are located at the vertices of the BC patch.

The physical-space normal vectors of the BC patch may be described by **inward_normal_list**; these are located at vertices, consistent with **point_range** and **point_list**. **inward_normal_list** is specified as an optional attribute because it is not always needed to enforce boundary-conditions, and the physical-space normals of a BC patch can usually be constructed from the grid. However, there are some situations, such as grid-coordinate singularity lines, where **inward_normal_list** becomes a required attribute because the normals cannot be generated from other information.

data_sets: is a list of boundary-condition data sets. In general, the proper **bc_data_set** instance(s) to impose on the BC patch is determined by the value of **the_type**.

For a few boundary-conditions, such as a symmetry plane or polar singularity, the value of **the_type** completely describes the equations to impose, and no instances of **bc_data_set** are needed. For 'simple' boundary-conditions, where a single set of Dirichlet and/or Neumann data

is applied a single **bc_data_set** will likely be used (althought this is not a requirement). For 'compound' boundary-conditions, where the equations to impose are dependent on local flow conditions, several instances of **bc_data_set** will likely be used.

refstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

refstate: is reference data applicable to the conditions of the BC patch.

class: is the class of data;

units: is the system of units;

dimension: The number of indices required to reference a node.

physical_dimension: The number of coordinates required to define a node position.

vertex_list_length: is the number of vertices making up the BC patch. If **point_range** is specified, then the value may be determined from the number of grid points (inclusive) between the beginning and ending indices of **point_range**. If **point_list** is specified then the value is a user input. **vertex_list_length** is also the number of elements in the list **inward_normal_list**.

face_center_list_length: is the number of cell faces making up the BC patch. If **point_range** has a value, then the value of **face_center_list_length** can be easily determined. If the BC patch is not logically rectangular (i.e., if **point_list** is specified), then the value of **face_center_list_length** cannot be determined and has to be a user input.

zonebc: is the calling **zone_bc**.

Formal propositions:

wr1: One and only one of the following attributes shall have a value: **point_range**, **point_list**, **elements**, **element_range**, **element_list**. but not both.

wr2: If **point_range** has a value it shall have the given value for **dimension**.

wr3: If **point_list** has a value, then it shall have the given values of **Int**, **dimension** and **VertexListLength**.

wr4: If **inward_normal_list** has a value, then it shall have the given values of **Float**, **dimension** and **VertexListLength**.

wr5: If **BCDataSets** has a value then **face_center_list_length** shall also have a value.

Informal propositions:

ip1: **InwardNormalIndex** shall have a single nonzero entry;

ip2: If **point_range** and **inward_normal_list** are specified, then a an ordering convention is needed for indices on the BC patch. An ordering convention is also needed if **point_range** is

specified and local data is present in the **bc_data_set** substructures. FORTRAN multidimensional array ordering shall be used.

5.2.2.3.9 bc_data_set

bc_data_set contains Dirichlet and Neumann data for a single set of boundary-condition equations. Its intended use is for simple boundary-condition types, where the equations imposed do not depend on local flow conditions.

Boundary-condition data is seperated by equation type into Dirichlet and Neumann conditions. Dirichlet boundary-conditions impose the value of the given variables, whereas Neumann boundary-conditions impose the normal derivative of the given variables.

The **bc** structure (clause 5.2.2.3.8) allows for an arbitrary list of boundary-condition data sets, described by the **bc_data_set** structure. For simple boundary-conditions, a single data set must be chosen from a list that may contain more than one element. Likewise, for a compound boundary-condition, a limited number of data sets must be chosen and applied appropriately. The mechanism for proper choice of data sets is controlled by the **the_type** attribute of the **bc** structure, the **the_type** attribute of the **bc_data_set** structure, and the boundary-condition type association table (Table 12).

bc is used for both simple and compound boundary-conditions; hence, the attribute **bc.the_type** is of type **bc_type**. Conversely, the structure **bc_data_set** is intended to enforce a single set of boundary-condition equations independent of local flow conditions (i.e., it is appropriate only for simple boundary-conditions). That is why the attribute **bc_data_set.simple_type** is of type **bc_type_simple** and not **bc_type**. The appropriate choice of data sets is determined by matching the value of **bc.the_type** with the value of **bc_data_set.simple_type** as specified in Table 12.

Although the model has a strict division between the two categories of boundary-condition types, in practice some overlap may exist. For example, some of the more general boundary-condition types, such as **bc_wall**, may include a situation of inflow/outflow (for instance if the wall is porous). These complications require further guidelines on appropriate definition and use of boundary-condition types. The real distinctions between **bc_type_simple** and **bc_type_compound** are as follows:

— **bc_type_simple** identifiers always match themselves; **bc_type_compound** never match themselves.

— **bc_type_simple** identifiers always produce a single match; **bc_type_compound** produce multiple matches.

— The usage rule for **bc_type_simple** identifiers is always trivial — apply the single matching data set regardless of local flow conditions.

Therefore, any boundary-condition that involves application of different data sets depending on

Table 12 – Associated boundary-condition types and usage rules

bc_type Identifier	**Associated bc_type_simple identifiers and usage rules**
bc_inflow	**bc_inflow_supersonic** **bc_inflow_subsonic** *usage rule:* if supersonic normal Mach, choose **bc_inflow_supersonic**, else choose **bc_inflow_subsonic**.
bc_outflow	**bc_outflow_supersonic** **bc_outflow_subsonic** *usage rule:* if supersonic normal Mach, choose **bc_outflow_supersonic**, else choose **bc_outflow_subsonic**.
bc_farfield	**bc_inflow_supersonic** **bc_inflow_subsonic** **bc_outflow_supersonic** **bc_outflow_subsonic** *usage rule:* if inflow and supersonic normal Mach, choose **bc_inflow_supersonic**, else if inflow, choose **bc_inflow_subsonic**, else if outflow and supersonic normal Mach, choose **bc_outflow_supersonic**, else, choose **bc_outflow_subsonic**.
bc_inflow_supersonic	**bc_inflow_supersonic** **bc_Dirichlet** *usage rule:* choose either; **bc_inflow_supersonic** takes precedence.
bc_outflow_supersonic	**bc_outflow_supersonic** **bc_Extrapolate** *usage rule:* choose either; **bc_outflow_supersonic** takes precedence.
all others	self-matching

local flow conditions should be classified as **bc_type_compound**.

NOTE 1 If a type that is classified **bc_type_simple** is desired to be used as a compound (**bc_wall** for a porous wall is an example), then it should somehow be reclassified. One option is to define a new **bc_type_compound** identifier and provide associated **bc_type_simple** types and a usage rule. Another option may be to allow some identifiers to be both **bc_type_simple** and **bc_type_compound** and let their appropriate use be based on context. This is still evolving.

For a given simple boundary-condition (i.e., one that is not dependent on flow conditions), the database provides a set of boundary-condition equations to be enforced through the definitions of **bc_data_set** and **bc_data**. Apart from the boundary-condition type, the precise equations to be enforced are described by boundary-condition solution data. These specified solution data are arranged by 'equation type':

Dirichlet: $Q = (Q)_{\text{specified}}$

Neumann: $\partial Q/\partial n = (\partial Q/\partial n)_{\text{specified}}$

The **Dirichlet_data** and **Neumann_data** attributes (of type **bc_data**) list both the solution variables involved in the variables (through the data-name conventions of clause G.3.3.12) and the specified solution data.

Two issues need to be addressed for specifying Dirichlet or Neumann boundary-condition data. The first is whether the data is global or local.

NOTE 2

global BC data: data applied globally to the BC patch; for example, specifying a uniform total pressure at an inflow boundary.

local BC data: data applied at each grid point of the BC patch; an example of this is varying total pressure specified at each vertex of the BC patch.

The second issue is describing the actual solution quantities that are to be specified. Both of these issues are addressed by use of the **data_array** structure.

For some types of boundary-conditions, many different combinations of solution quantities could be specified. For example, **bc_inflow_subsonic** requires four solution quantities to be specified in 3-D, but what those four quantities are varies with applications (e.g., internal versus external flows) and codes. The actual data being specified for any **bc_type** is given by the list of **data_array** instances included in the **Dirichlet_data** and **Neumann_data** attributes (actually by the identifier attached to each instance of **data_array**). This reduces the potential problem of having to specify *many* versions of a given **bc_type** (e.g., **bc_inflow_subsonic1**, **bc_inflow_subsonic2** etc.), where each has a precisely defined set of Dirichlet data. Instead, only the number of Dirichlet or Neumann quantities must be provided for each **bc_type**.

The global versus local issue can easily be handled by storing a scalar for the global BC data case, and storing an array for the local BC data case.

By convention, if the **Dirichlet_data** and **Neumann_data** are not present in an instance of **bc_data_set**, then application codes (e.g., flow solvers) are free to enforce appropriate boundary-conditions for the given type of **bc_type_simple**. Furthermore, if insufficient data is present (e.g., only one Dirichlet variable is present for a subsonic inflow condition), then application codes are free to fill out the boundary-condition data as appropriate for the **bc_type_simple** identifier.

To facilitate implementation of boundary-conditions into existing flow solvers, if no boundary-condition data is specified, then flow solvers are free to enforce any appropriate boundary-condition equations. This includes situations where instances of **bc_data_set**, **bc_data** or **data_array** are absent within the boundary-condition hierarchy. In this case the **reference_state** specifies the reference-state conditions from which the flow solver should extract the boundary-condition data. Within the boundary-condition hierarchy, **reference_state** instances may be present at any of the **zone_bc**, **bc** or **bc_data_set** levels with the lowest taking precedence.

EXPRESS specification:

```
*)
ENTITY bc_data_set;
  description    : OPTIONAL texts;
  simple_type    : bc_type_simple;
  gridloc        : OPTIONAL grid_location;
  Dirichlet_data : OPTIONAL bc_data;
  Neumann_data   : OPTIONAL bc_data;
  rstate         : OPTIONAL reference_state;
  dclass         : OPTIONAL data_class;
  dimunits       : OPTIONAL dimensional_units;
DERIVE
  location                : grid_location := NVL(gridloc, vertex);
  refstate                : reference_state := NVL(rstate, bc.refstate);
  class                   : data_class := NVL(dclass, bc.class);
  units                   : dimensional_units := NVL(dimunits, bc.units);
  vertex_list_length      : positive := bc.vertex_list_length;
  face_center_list_length : positive := bc.face_center_list_length;
  data_array_length       : INTEGER := BCDataSet_DataArrayLength(SELF);
INVERSE
  bc : bc FOR data_sets;
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

simple_type: is the boundary-condition type, which gives general information on the boundary-condition equations to be enforced.

gridloc: is non-default location information;

location: is the location of local data arrays (if any) provided in **Dirichlet_data** and **Neumann_data**. Local boundary-condition data may be defined either at vertices or boundary face centers.

Dirichlet_data: is boundary-condition data for Dirichlet conditions which may be constant over the BC patch or defined locally at each point of the patch.

Neumann_data: is boundary-condition data for Neumann conditions which may be constant over the BC patch or defined locally at each point of the patch.

rstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

refstate: is reference quantities applicable to the set of boundary-condition data.

class: is the class of data contained in the boundary-condition data.

units: is the system of units employed.

vertex_list_length: is the number of vertices in the BC patch.

face_center_list_length: is the number of cell faces in the BC patch.

data_array_length: is the length of the data arrays.

bc: is the calling **bc**.

5.2.2.3.10 bc_data

bc_data contains a list of variables and associated data for boundary-condition specification. Each variable may be given as global data (i.e., a scalar) or local data defined at each grid point of the BC patch. By convention all data specified in a given instance of **bc_data** is to be used in the same *type* of boundary-condition equation.

EXAMPLE 1 The Dirichlet and Neumann conditions in **bc_data_set** use seperate **bc_data** structures.

This structure allows a given instance of **bc_data** to have a mixture of global and local data.

EXAMPLE 2 If the Dirichlet condition consists of a uniform stagnation pressure but with with a non-uniform velocity profile, then the stagnation pressure can be described by a scalar in the **data_global** list and the velocity by an array in the **data_local** list.

EXPRESS specification:

```
*)
ENTITY bc_data;
```



```
  description : OPTIONAL texts;
  data_global : LIST OF data_array;
  data_local  : LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  class             : data_class := BCData_class(SELF);
  units             : dimensional_units := BCData_DimUnits(SELF);
  data_array_length : INTEGER := NVL(DataSetD[1].DataArrayLength, 0) +
                                 NVL(DataSetN[1].DataArrayLength, 0);
INVERSE
  Dirichlet : BAG [0:1] OF bc_data_set FOR Dirichlet_data;
  Neumann   : BAG [0:1] OF bc_data_set FOR Neumann_data;
WHERE
  wr1 : SIZEOF(DataSetD+DataSetN) = 1;
  wr4 : SIZEOF(QUERY(v <* data_global |
               consistent_data_array(v, 1, [1],
                                     class, units, ?, ?))
        = SIZEOF(data_global);
  wr5 : SIZEOF(QUERY(v <* data_local |
               consistent_data_array(v, 1, [1],
                                     class, units, ?, ?))
        = SIZEOF(data_local);
--  wr4 : DataArraysOK(DataGlobal, DataTypeG, 1, 1);
--  wr5 : DataArraysOK(DataLocal, DataTypeL, 1, DataArrayLength);
END_ENTITY;
(*
```

Attribute definitions:

description: is annotation;

data_global: is global data;

data_local: is local data;

dclass: non-default data class;

dimunits: non-default system of units;

class: is the class of data;

units: is the system of units for the data;

data_array_length: is the length data arrays;

Dirichlet: is the calling **bc_data_set** which uses this **bc_data** for Dirichlet data;

Neumann: is the calling **bc_data_set** which uses this **bc_data** for Neumann data;

Formal propositions:

wr1: An instance of **bc_data** shall be called in the role of either **Dirichlet_data** or **Neumann_data**.

5.2.2.3.11 family

EXPRESS specification:

```
*)
ENTITY family;
  description : OPTIONAL texts;
  conditions  : LIST OF bc_type;
  geometry    : LIST OF externally_defined_item;
END_ENTITY;
(*
```

Attribute definitions:

description: is annotation;

conditions: the family's boundary condition types;

geometry: the family's geometric information;

5.2.2.3.12 flow_equation_set

flow_equation_set is a general description of governing flow equations. It includes the dimensionality of the governing equations.

EXPRESS specification:

```
*)
ENTITY flow_equation_set;
  description          : OPTIONAL texts;
  equation_dimension   : positive;
  equations            : OPTIONAL governing_equations;
  state                : OPTIONAL gas_model;
  viscosity            : OPTIONAL viscosity_model;
  thermal_conductivity : OPTIONAL thermal_conductivity_model;
  closure              : OPTIONAL turbulence_closure;
  turbulence           : OPTIONAL turbulence_model;
  dclass               : OPTIONAL data_class;
  dimunits             : OPTIONAL dimensional_units;
DERIVE
  dimension : positive := NVL(base[1].dimension,0) +
                          NVL(zone[1].dimension,0);
```



```
  class     : data_class :=
                    NVL(dclass, inherit_class_from_base_zone(base, zone));
  units     : dimensional_units :=
                    NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_base FOR equations;
  zone : BAG [0:1] OF zone FOR equations;
WHERE
  wr1 : SIZEOF(base) + SIZEOF(zone) = 1;
END_ENTITY;
(*
```

Attribute definitions:

description: is annotation;

equation_dimension: is the dimensionality of the governing equations; it is the number of spatial variables describing the flow.

equations: describes the general class of equations.

state: describes the equation of state.

viscosity: describes the auxiliary relations for molecular viscosity.

thermal_conductivity: describes the auxiliary relations for the thermal conductivity coefficient.

closure: describes the turbulent closure for Reynolds-averaged Navier-Stokes equations.

turbulence: describes the turbulence model for Reynolds-averaged Navier-Stokes equations.

dclass: non-default class of data;

dimunits: non-default system of units;

class: is the class of data contained in the **flow_equation_set**.

units: is the system of units.

dimension: The number of indices required to reference a node.

base: is the calling calling **cfd_base**;

zone: is the calling calling **zone**;

Formal propositions:

wr1: A **flow_equation_set** shall be called by either a **cfd_base** or by a **zone**.

5.2.2.3.13 governing_equations

governing_equations describes the class of governing flow equations associated with the solution.

EXPRESS specification:

```
*)
ENTITY governing_equations;
  description     : OPTIONAL texts;
  equation_type   : governing_equations_type;
  diffusion_model : OPTIONAL ARRAY [1:diff] OF BOOLEAN;
DERIVE
  dimension : positive := equation_set.dimension;
  diff      : INTEGER := (dimension**2 + dimension)/2;
INVERSE
  equation_set : flow_equation_set FOR equations;
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

equation_type: is the kind of equation;

diffusion_model: describes the viscous diffusion terms modelled in the flow equations, and is applicable only to Navier-Stokes equations. Typically, thin-layer approximations include only the diffusion terms in one or two computational-coordinate directions. **diffusion_model** encodes the coordinate directions that include second-derivative and cross-derivative diffusion terms. The first **dimension** elements are second-derivative terms and the remainder elements are cross-derivative terms. A value of TRUE indicates the diffusion term is modelled, and FALSE indicates that it is not modelled. In 3–D, the encoding of **diffusion_model** is as follows:

element	modelled terms
$n = 1$	diffusion terms in i ($\partial^2/\partial\xi^2$)
$n = 2$	diffusion terms in j ($\partial^2/\partial\eta^2$)
$n = 3$	diffusion terms in k ($\partial^2/\partial\zeta^2$)
$n = 4$	cross-diffusion terms in i-j ($\partial^2/\partial\xi\partial\eta$ and $\partial^2/\partial\eta\partial\xi$)
$n = 5$	cross-diffusion terms in j-k ($\partial^2/\partial\eta\partial\zeta$ and $\partial^2/\partial\zeta\partial\eta$)
$n = 6$	cross-diffusion terms in k-i ($\partial^2/\partial\zeta\partial\xi$ and $\partial^2/\partial\xi\partial\zeta$)

where derivatives in the i, j and k computational-coordinates are ξ, η and ζ, respectively.

EXAMPLE 1 The full Navier-Stokes equations in 3–D are indicated by:
diffusion_model = `[TRUE,TRUE,TRUE,TRUE,TRUE,TRUE]` while the thin-layer equations including only diffusion in the j-direction are indicated by:
diffusion_model = `[FALSE,TRUE,FALSE,FALSE,FALSE,FALSE]`.

dimension: The number of indices required to reference a node.

diff: is the number of elements in **diffusion_model**. For 1-D this is one, for 2-D it is three, and for 3-D it is six.

equation_set: is the calling **flow_equation_set**.

5.2.2.3.14 fd_model

EXPRESS specification:

```
*)
ENTITY fd_model;
  description : OPTIONAL texts;
  data        : LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_fd_model FOR fd_model;
  ABSTRACT SUPERTYPE;
  ONEOF(gas_model,
        thermal_conductivity_model,
        turbulence_closure,
        turbulence_model,
        viscosity_model);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

description: is annotations;

data: is the data;

dclass: is non-default class of data;

dimunits: is non-default dimensional units.

5.2.2.3.15 gas_model

gas_model describes the equation of state model used in the governing equations to relate pressure, temperature and density.

EXPRESS specification:

```
*)
ENTITY gas_model
```



```
  SUBTYPE OF (fd_model);
  model_type : gas_model_type;
DERIVE
  class : data_class :=
                    NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                    NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR state;
WHERE
  wr1 : SIZEOF(QUERY(v <* data |
              consistent_data_array(v, 1, [1],
                                    class, units, ?, ?))
        = SIZEOF(data);
--  wr1 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation of state model.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

Formal propositions:

wr1: The inherited **DataArrays** arrays shall be consistent.

5.2.2.3.16 thermal_conductivity_model

thermal_conductivity_model describes the model for relating the thermal-conductivity coefficient (k) to temperature.

EXPRESS specification:

```
*)
ENTITY thermal_conductivity_model
  SUBTYPE OF (fd_model);
  model_type : thermal_conductivity_model_type;
DERIVE
  class : data_class :=
                    NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
```



```
                    NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR thermal_conductivity;
WHERE
  wr1 : SIZEOF(QUERY(v <* data |
              consistent_data_array(v, 1, [1],
                                    class, units, ?, ?))
        = SIZEOF(data);
--  wr1 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular model type.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

Formal propositions:

wr1: The inherited **DataArrays** arrays shall be consistent.

5.2.2.3.17 turbulence_closure

turbulence_closure describes the turbulence closure for the Reynolds stress terms of the Navier-Stokes equations.

EXPRESS specification:

```
*)
ENTITY turbulence_closure
  SUBTYPE OF (fd_model);
  closure_type : turbulence_closure_type;
DERIVE
  class : data_class :=
                    NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                    NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR closure;
WHERE
  wr1 : SIZEOF(QUERY(v <* data |
              consistent_data_array(v, 1, [1],
```



```
                                    class, units, ?, ?))
        = SIZEOF(data);
--  wr1 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

closure_type: is the particular closure type.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

Formal propositions:

wr1: The inherited **DataArrays** arrays shall be consistent.

5.2.2.3.18 turbulence_model

turbulence_model describes the equation set used to model the turbulence quantities.

.

EXPRESS specification:

```
*)
ENTITY turbulence_model
  SUBTYPE OF (fd_model);
  model_type      : turbulence_model_type;
  diffusion_model : OPTIONAL ARRAY [1:diff] OF BOOLEAN;
DERIVE
  class     : data_class :=
                    NVL(SELF\fd_model.dclass, equation_set.class);
  units     : dimensional_units :=
                    NVL(SELF\fd_model.dimunits, equation_set.units);
  dimension : positive := equation_set.dimension;
  diff      : INTEGER := (dimension**2 + dimension)/2;
INVERSE
  equation_set : flow_equation_set FOR turbulence;
WHERE
  wr1 : SIZEOF(QUERY(v <* data |
              consistent_data_array(v, 1, [1],
                                    class, units, ?, ?))
        = SIZEOF(data);
```



```
--  wr1 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation set;

diffusion_model: is the description of the viscous diffusion terms included in the turbulent transport model equations;

class: is the class of data;

units: is the dimensional units;

dimension: The number of indices required to reference a node.

diff: is the number of elements in **diffusion_model**. For 1-D this is one, for 2-D it is three, and for 3-D it is six.

equation_set: is the calling **flow_equation_set**.

Formal propositions:

wr1: The inherited **DataArrays** arrays shall be consistent.

5.2.2.3.19 viscosity_model

viscosity_model describes the model for relating molecular viscosity (μ) to temperature.

EXPRESS specification:

```
*)
ENTITY viscosity_model
  SUBTYPE OF (fd_model);
  model_type : viscosity_model_type;
DERIVE
  class : data_class :=
                    NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                    NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR viscosity;
WHERE
  wr1 : SIZEOF(QUERY(v <* data |
              consistent_data_array(v, 1, [1],
                                    class, units, ?, ?))
        = SIZEOF(data);
```



```
--  wr1 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation of state model.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

Formal propositions:

wr1: The inherited **DataArrays** arrays shall be consistent.

5.2.2.3.20 reference_state

reference_state describes a reference state, which is a list of geometric or flow-state quantities defined at a common location or condition. Examples of typical reference states associated with CFD calculations are freestream, plenum, stagntation, inlet and exit.

EXPRESS specification:

```
*)
ENTITY reference_state;
  description       : OPTIONAL texts;
  state_description : OPTIONAL text;
  data              : LIST OF data_array;
  dclass            : OPTIONAL data_class;
  dimunits          : OPTIONAL dimensional_units;
DERIVE
  class : data_class :=
    NVL(dclass,
        inherited_class_for_refstate(data_set, bc, zone_bc, zone, base));
  units : dimensional_units :=
    NVL(dimunits,
        inherited_units_for_refstate(data_set, bc, zone_bc, zone, base));
INVERSE
  base     : BAG [0:1] OF cfd_base FOR refstate;
  zone     : BAG [0:1] OF zone FOR refstate;
  data_set : BAG [0:1] OF bc_data_set FOR refstate;
  bc       : BAG [0:1] OF bc FOR refstate;
  zone_bc  : BAG [0:1] OF zone_bc FOR refstate;
WHERE
```



```
  wr1 : SIZEOF(CGNSBase) + SIZEOF(Zone) + SIZEOF(BCDataSet) +
        SIZEOF(BC) + SIZEOF(ZoneBC) = 1;
  wr3 : SIZEOF(QUERY(v <* data |
                consistent_data_array(v, 1, [1],
                                      class, units, ?, ?))
        = SIZEOF(data);
--  wr3 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

state_description: is a description of the reference state;

data: is the reference state data;

dclass: non-default data class;

dimunits: non-default dimensional units;

class: is the class of data;

units: is the system of units;

Formal propositions:

wr1: A reference state shall be called by one and only one of: **cfd_base**, **zone**, **bc_data_set**, **bc**, **zone_bc**;

wr3: The data arrays shall be consistent.

5.2.2.3.21 convergence_history

Flow solver convergence history information is described by the **convergence_history** structure.

Measures used to record convergence vary greatly among current flow-solver implementations. Convergence information typically includes global forces, norms of equation residuals, and norms of solution changes.

NOTE 1 Attempts to systematically define a set of convergence measures have been futile. For global parameters, such as forces and moments, clause G.3.3.12 provides a set of standarized data-array identifiers. For equation residuals and solution changes, no such standard list exists. Therefore, **adhoc_data_name** has to be used as the data-array identifier. It is suggested that identifiers for norms of equation residuals begin with RSD, and those for solution changes begin with CHG. For example, **'RSDMassRMS'** could be used for the L_2-norm (RMS) of mass conservation residuals.

EXPRESS specification:

```
*)
ENTITY convergence_history;
  description      : OPTIONAL texts;
  norm_definitions : OPTIONAL text;
  iterations       : INTEGER;
  data             : LIST OF data_array;
  dclass           : OPTIONAL data_class;
  dimunits         : OPTIONAL dimensional_units;
DERIVE
  class : data_class :=
                     NVL(dclass, inherit_class_from_base_zone(base, zone));
  units : dimensional_units :=
                     NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_base FOR history;
  zone : BAG [0:1] OF zone FOR history;
WHERE
  wr1 : SIZEOF(CGNSBase) + SIZEOF(Zone) = 1;
  wr3 : SIZEOF(QUERY(v <* data |
               consistent_data_array(v, 1, [iterations],
                                     class, units, ?, ?))
        = SIZEOF(data);
--  wr3 : DataArraysOK(DataArrays, DataType, 1, [Iterations]);
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

norm_definitions: is a description of the convergence information recorded as **data**;

iterations: is the number of iterations for which convergence information is recorded;

data: is convergence history data;

dclass: non-default class of data;

dimunits: non-default system of dimensional units;

class: is the class of data;

units: is the system of dimensional units;

base: is the referencing database;

zone: is the referencing zone.

Formal propositions:

wr1: A **convergence_history** instance shall be called by either a CFD database or a zone;

wr3: The data arrays shall be consistent.

5.2.2.3.22 discrete_data

discrete_data provides a description of generic discrete data (i.e., data defined on a computational grid); it is identical to **flow_solution** except for its entity name. This structure can be used to store field data, such as fluxes or equation residuals, that is not typically considered part of the flow solution.

EXPRESS specification:

```
*)
ENTITY discrete_data;
  description : OPTIONAL texts;
  gridloc     : OPTIONAl grid_location;
  rind        : OPTIONAL rind;
  data        : LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  location    : grid_location := NVL(gridloc, vertex);
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  dimension   : INTEGER := zone.dimension;
  vertex_size : ARRAY [1:dimension] OF INTEGER := zone.vertex_size;
  cell_size   : ARRAY [1:dimension] OF INTEGER := zone.cell_size;
  field_size  : ARRAY [1:dimension] OF positive :=
                field_data_size(dimension, vertex_size, cell_size,
                                location, rind);
INVERSE
  zone : zone FOR field_data;
WHERE
  wr1 : (NOT EXISTS(rind)) XOR (rind.dimension = dimension);
  wr2 : NOT ((schdot+'UNSTRUCTURED_ZONE' IN TYPEOF(zone)) AND EXISTS(rind));
  wr3 : SIZEOF(QUERY(v <* data |
               consistent_data_array(v, 1, field_size,
                                     class, units, ?, ?))
        = SIZEOF(data);
--  wr3 : DataArraysOK(DataArrays, DataType, dimension,
--                     FieldDataSize(dimension, VertexSize, CellSize,
--                                   GridLocation, Rind));
  wr5 : (location = vertex) XOR (location = cell_center);
END_ENTITY;
(*
```


Attribute definitions:

description: is annotation;

gridloc: non-default grid location;

rind: is the Rind planes; if absent then it is equivalent to having an instance of **Rind** whose **RindPlanes** array contains all zeroes.

data: is the data;

dclass: non-default data class;

dimunits: non-default dimensional units;

location: is the location of data with respect to the grid;

class: is the class of data;

units: is the system of dimensional units;

dimension: The number of indices required to reference a node.

vertex_size: is the number of core vertices in each index direction;

cell_size: is the number of core cells in each index direction;

field_size: is the size of the discrete data arrays;

zone: is the calling zone.

Formal propositions:

wr1: If **rind** has a value then its **dimension** shall be the same as the the **discrete_data dimension**.

wr2: Discrete data for an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

wr3: The data arrays shall be consistent.

wr5: The grid location shall be either **vertex** or **cell_center**.

Informal propositions:

ip1: All data contained within this structure shall be defined at the same grid location and have the same amount of rind-point data.

5.2.2.3.23 integral_data

integral_data provides a description of generic global or integral data that may be associated with a particular zone or an entire database. In contrast to **discrete_data**, integral data is not associated with any specific field location.

EXPRESS specification:

```
*)
ENTITY integral_data;
  description : OPTIONAL LIST OF STRING;
  data        : LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  class        : data_class :=
                       NVL(dclass, inherit_class_from_base_zone(base, zone));
  units : dimensional_units :=
                       NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_base FOR miscellaneous;
  zone : BAG [0:1] OF zone FOR global_data;
WHERE
  wr1 : SIZEOF(CGNSBase) + SIZEOF(Zone) = 1;
  wr2 : (NOT EXISTS(DataArrays)) XOR
        (EXISTS(DataArrays) AND EXISTS(DataType));
  wr3 : SIZEOF(QUERY(v <* data |
                consistent_data_array(v, 1, [1],
                                      class, units, ?, ?))
        = SIZEOF(data);
--  wr3 : DataArraysOK(DataArrays, DataType, 1, [1]);
END_ENTITY;
(*
```

Attribute definitions:

description: is annotations;

data: is the data;

dclass: non-default class of data;

dimunits: non-default system of units;

class: is the class of data;

units: is the system of units;

base: is the calling **cfd_base**.

zone: is the calling **zone**.

Formal propositions:

wr1: An instance of **integral_data** shall be referenced by either a **cfd_base** or a **zone**;

wr2: If **DataArrays** has a value then **DataType** shall also have a value;

wr3: The data arrays shall be consistent.

5.2.2.4 Fluid dynamics data function definitions

5.2.2.4.1 schname

The function **schname** returns a string consisting of the uppercase name of the schema.

EXPRESS specification:

```
*)
FUNCTION schname : STRING;
  RETURN('CFD_AIM');
END_FUNCTION;
(*
```

Argument definitions:

RETURNS: A string literal consisting of the name of the schema, in uppercase.

5.2.2.4.2 schdot

The function **schdot** returns a string consisting of the uppercase name of the schema with an appended period.

EXPRESS specification:

```
*)
FUNCTION schdot : STRING;
  RETURN(schname+'.');
END_FUNCTION;
(*
```

Argument definitions:

RETURNS: A string literal consisting of the name of the schema, in uppercase, appended with a period.

5.2.2.4.3 consistent_data_array

The function **consistent_data_array** returns TRUE if the attribute values of a **data_array** are consistent with an identified set of values.

EXPRESS specification:

```
*)
FUNCTION consistent_data_array
  (da    : data_array;
   dim   : INTEGER;
   szs   : ARRAY OF INTEGER;
   id    : data_name;
   class : data_class;
   units : dimensional_units;
   dexp  : dimensional_exponents;
   conv  : data_conversion) : BOOLEAN;
  IF (NOT EXISTS(da)) THEN RETURN(FALSE); END_IF;
  IF (EXISTS(dim)   AND (da.dimension <> dim))   THEN RETURN(FALSE); END_IF;
  IF (EXISTS(id)    AND (da.identifier <> id))   THEN RETURN(FALSE); END_IF;
  IF (EXISTS(class) AND (da.class <> class))     THEN RETURN(FALSE); END_IF;
  IF (EXISTS(units) AND (da.units <> units))     THEN RETURN(FALSE); END_IF;
  IF (EXISTS(dexp)  AND (da.exponents <> dexp))  THEN RETURN(FALSE); END_IF;
  IF (EXISTS(conv)  AND (da.conversion <> conv)) THEN RETURN(FALSE); END_IF;
  RETURN(TRUE);
END_FUNCTION;
(*
```

Argument definitions:

da: A **data_array**;

dim: A possible value for **da**'s **dimension** attribute;

id: A possible value for **da**'s **identifier** attribute;

class: A possible value for **da**'s **class** attribute;

units: A possible value for **da**'s **units** attribute;

dexp: A possible value for **da**'s **exponents** attribute;

conv: A possible value for **da**'s **conversion** attribute;

RETURNS: FALSE if there is no value for **da**; if there is a value for **da**, FLASE is returned if any of the possible attributes has a value and it is not the same as the actual attribute value. Otherwise, TRUE is returned.

5.2.2.4.4 inherit_class_from_base_zone

Given either a aggregate of **cfd_base** or of **zone** the function returns the **class**.

EXPRESS specification:

```
*)
FUNCTION inherit_class_from_base_zone(base : AGGREGATE OF cfd_base;
                                      zone : AGGREGATE OF zone) : data_class;
  IF (SIZEOF(base) > 0) THEN
    RETURN(base[1].class);
  ELSE
    IF (SIZEOF(zone) > 0) THEN
      RETURN(zone[1].class);
    END_IF;
  END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

base: A possibly empty aggregate of **cfd_base**.

zone: A possibly empty aggregate of **zone**

RETURNS: The value of the **class** attribute for the first **cfd_base** of the first **zone**. If there is an error in the arguments, indeterminate is returned.

5.2.2.4.5 inherit_units_from_base_zone

Given either an aggregate of **cfd_base** or of **zone** the function returns the **units**.

EXPRESS specification:

```
*)
FUNCTION inherit_units_from_base_zone(base : AGGREGATE OF cfd_base;
                                      zone : AGGREGATE OF zone) : dimensional_units;
  IF (SIZEOF(base) > 0) THEN
    RETURN(base[1].units);
  ELSE
    IF (SIZEOF(zone) > 0) THEN
      RETURN(zone[1].units);
    END_IF;
  END_IF;
  RETURN(?);
END_FUNCTION;
(*
```


Argument definitions:

base: A possibly empty aggregate of **cfd_base**.

zone: A possibly empty aggregate of **zone**

RETURNS: The value of the **units** attribute for the first **cfd_base** of the first **zone**. If there is an error in the arguments, indeterminate is returned.

5.2.2.4.6 inherited_class_for_refstate

The **inherited_class_for_refstate** function determines a class of data.

EXPRESS specification:

```
*)
FUNCTION inherited_units_for_refstate(bd : AGGREGATE OF bc_data_set;
                                      bc : AGGREGATE OF bc;
                                      zb : AGGREGATE OF zone_bc;
                                      zn : AGGREGATE OF zone;
                                      db : AGGREGATE OF cfd_base)
         : data_class;
  IF (SIZEOF(bd) > 0) THEN RETURN(bd[1].class); END_IF;
  IF (SIZEOF(bc) > 0) THEN RETURN(bc[1].class); END_IF;
  IF (SIZEOF(zb) > 0) THEN RETURN(zb[1].class); END_IF;
  IF (SIZEOF(zn) > 0) THEN RETURN(zn[1].class); END_IF;
  IF (SIZEOF(db) > 0) THEN RETURN(db[1].class); END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

bd: A possibly empty aggregate of **bc_data_set**;

bc: A possibly empty aggregate of **bc**;

zb: A possibly empty aggregate of **zone_bc**;

zn: A possibly empty aggregate of **zone**;

db: A possibly empty aggregate of **cfd_base**;

RETURNS: The value of the **data_class** which has the highest precedence. If there is an error in the arguments, indeterminate is returned.

5.2.2.4.7 inherited_units_for_refstate

The **inherited_units_for_refstate** function determines dimensional units.

EXPRESS specification:

```
*)
FUNCTION inherited_units_for_refstate(bd : AGGREGATE OF bc_data_set;
                                      bc : AGGREGATE OF bc;
                                      zb : AGGREGATE OF zone_bc;
                                      zn : AGGREGATE OF zone;
                                      db : AGGREGATE OF cfd_base)
         : dimensional_units;
  IF (SIZEOF(bd) > 0) THEN RETURN(bd[1].units); END_IF;
  IF (SIZEOF(bc) > 0) THEN RETURN(bc[1].units); END_IF;
  IF (SIZEOF(zb) > 0) THEN RETURN(zb[1].units); END_IF;
  IF (SIZEOF(zn) > 0) THEN RETURN(zn[1].units); END_IF;
  IF (SIZEOF(db) > 0) THEN RETURN(db[1].units); END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

bd: A possibly empty aggregate of **bc_data_set**;

bc: A possibly empty aggregate of **bc**;

zb: A possibly empty aggregate of **zone_bc**;

zn: A possibly empty aggregate of **zone**;

db: A possibly empty aggregate of **cfd_base**;

RETURNS: The value of the **dimensional_units** which has the highest precedence. If there is an error in the arguments, indeterminate is returned.

5.2.2.4.8 grid_data_size

The **grid_data_size** function calculates the required array sizes of grid-coordinate data for **grid_coordinates**, namely the core points plus any rind points.

EXPRESS specification:

```
*)
FUNCTION grid_data_size(arg : grid_coordinates) : ARRAY OF positive;
LOCAL
  dim    : positive := arg.dimension;
  result : ARRAY [1:dim] OF positive;
END_LOCAL;
  REPEAT i := 1 TO dim;
    result[i] := arg.vertex_size[i];
  END_REPEAT;
```



```
  IF (EXISTS(arg.rind)) THEN
    REPEAT i := 1 TO dim;
      result[i] := result[i] + arg.rind.planes[2*i-1]
                             + arg.rind.planes[2*i];
    END_REPEAT;
  END_IF;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **grid_coordinates**

RETURNS: An array of **positive** of size **arg.dimension**.

5.2.2.4.9 field_data_size

The **field_data_size** function calculates data array sizes for data defined on a grid, namely the core points plus any rind points. The actual values also depend on the grid location.

EXPRESS specification:

```
*)
FUNCTION field_data_size(dim  : positive;
                         vs   : ARRAY OF positive;
                         cs   : ARRAY OF positive;
                         gl   : grid_location;
                         rind : rind)  : ARRAY OF nonnegative;
LOCAL
  result : ARRAY [1:dim] OF positive;
END_LOCAL;
  IF (gl = vertex) THEN
    REPEAT i := 1 TO SIZEOF(vs);
      result[i] := vs[i];
    END_REPEAT;
  ELSE
    IF (gl = cell_center) THEN
      REPEAT i := 1 TO SIZEOF(cs);
        result[i] := cs[i];
      END_REPEAT;
    END_IF;
  END_IF;
  IF (EXISTS(rind)) THEN
    REPEAT i := 1 TO dim;
      result[i] := result[i] + rind.planes[2*i-1]
                             + rind.planes[2*i];
    END_REPEAT;
```



```
  END_IF;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

dim: A **dimension**

vs: An array of **positive** of size **dim** containing the number of core vertices in each index direction.

cs: An array of **positive** of size **dim** containing the number of core cells in each index direction.

gl: A **grid_location** specifying the location of data with respect to a grid.

rind: A **rind** specifying the number of rind planes included in the data.

RETURNS: An array of **positive** of size **dim**.

EXPRESS specification:

```
*)
END_SCHEMA; -- cfd_aim
(*
```


6 Conformance requirements

Annex A
(normative)
AIM EXPRESS expanded listing

The following EXPRESS is the expanded form of the short form schema given in 5.2. In the event of any discrepancy between the short form and this expanded listing, the expanded listing shall be used.

Annex B
(normative)
AIM short names

Table B.1 provides the short names of entities specified in the AIM of this part of ISO 10303. Requirements on the use of the short names are found in the implementation methods included in ISO 10303.

Annex C
(normative)
Implementation method specific requirements

The implementation method defines what types of exchange behaviour are required with respect to this part of ISO 10303. Conformance to this part of ISO 10303 shall be realized in an exchange structure. The file format shall be encoded according to the syntax and EXPRESS language mapping defined in ISO 10303-21 and in the AIM defined in annex A of this part of ISO 10303. The header of the exchange structure shall identify use of this part of ISO 10303 by the schema name '(TBD — SCHEMA NAME)'.

Annex D
(normative)
Protocol Implementation Conformance Statement (PICS) proforma

This clause lists the optional elements of this part of ISO 10303. An implementation may choose to support any combination of these optional elements. However, certain combinations of options are likely to be implemented together. These combinations are called conformance classes and are described in the subclauses of this annex.

This annex is in the form of a questionnaire. This questionnaire is intended to be filled out by the implementor and may be used in preparation for conformance testing by a testing laboratory. The completed PICS proforma is referred to as a PICS.

Annex E
(normative)
Information object registration

Annex F
(informative)
Application activity model

The application activity model (AAM) is provided as an aid in understanding the scope and information requirements defined in this application protocol. The model is presented as a set of figures that contain the activity diagrams and a set of definitions of the activities and their data. The application activity model is given in Figures F.1 to F.6. Activities and data flows that are out of scope are marked with an asterisk.

F.1 Introduction

The application activity model (AAM) is provided as an aid to understanding the scope and information requirements defined in this application protocol. At present, this AAM is restricted in scope to data associated with the process and activities of Computational Fluid Dynamics (CFD) analysis. In the future, the AAM will be extended to include aerodynamic data from ground test and flight test sources.

F.2 Application activity model background

This model describes the overall CFD analysis process, from the perspective of an aerospace manufacturer. The fluid dynamics AP will be developed in stages. The reader is referred to clause 1 for a full description of the initial and ultimate scope of this standard. In its initial state, this part of ISO 10303 defines data standards for laminar, transitional, and turbulent flow of a homogeneous ideal gas. Analyses may be performed on any combination of multi-block structured and/or unstructured grids.

The initial purpose of this part of ISO 10303 is to provide a standard for recording and recovering computer data associated with the numerical solution of the equations of fluid dynamics. The format implemented by this standard is (1) general, (2) portable, (3) expandable, and (4) durable. It is expected that this standard will be extended, in future activity, to provide for storage and retrieval of data from non-analytical sources such as wind tunnel or water tank testing, and flight testing or sea trials.

This part of ISO 10303 consists of a collection of conventions for the storage and retrieval of CFD (computational fluid dynamics) data. The system provides for a standard format for recording the data. A key characteristic of CFD data is that is consists of a relatively small number of very large data arrays. The data format is a conceptual entity established by the documentation intended to do the following:

— Facilitate the exchange of CFD data

  - between sites.
  - between applications codes.

- across computing platforms.

— Stabilize the archiving of CFD data.

The conventions of this part of ISO 10303 provide for recording a complete and flexible problem description. The exact meaning of a subsonic inflow boundary condition, for example, can be described in complete detail if desired. User comments can be appended nearly anywhere, affording the opportunity, for instance, for date stamping or history information to be included. Dimension and sizing information is carefully defined. Any number of flow variables may be recorded, with or without standard names, and it is also possible to add user-defined or site-specific data. These features afford the opportunity for applications to perform extensive error checking if desired.

Because of this generality, this part of ISO 10303 provides for the recording of much more descriptive information than current applications normally use. However, the provisions for this data are layered so that much of it is optional. It should be practical to convert most current applications to conform to this part of ISO 10303 with little or no conceptual change, retaining the option to take advantage of more detailed descriptions as that becomes desirable.

The specifications of this part of ISO 10303 currently cover the bulk of CFD data that one might wish to exchange among sites or applications; for instance, nearly any type of field data can be recorded, and, based on its name, found and understood by any code that needs it. Global data (e.g., freestream Mach Number, Reynolds number, angle of attack) and physical modeling instructions (e.g., thin layer assumptions, turbulence model) may be specified.

Nevertheless, there are items specific to individual applications for which there is currently no specification within this part of ISO 10303. Most commonly, these are operational instructions, such as number of sweeps, solution method, multigrid directives, and so on. Owing to the miscellaneous nature of this data, there has been no attempt to codify it within a global standard. It is therefore expected that many applications will continue to require small user-generated input files, presumably in ASCII format.

This part of ISO 10303 provides for an extensive set of CFD data. Most applications will make use of only a small subset of this data. Further, inasmuch as applications are viewed as editors that are in the process of building the database, most of them are intended for use on incomplete data sets. Therefore, it is not required that all the data elements specified by these conventions be complete in order for a database to be compliant with this part of ISO 10303. The user must ensure that the current state of the database will support whatever application may be launched. Of course, the application should gracefully handle any absence or deficiency of data. The validity, accuracy and completeness of the data are determined entirely by the applications software.

This part of ISO 10303 serves not only to facilitate the mapping of data onto the compliant file structure but also to standardize the meaning of the recorded data. Thus there are two kinds of conventions operative this standard.

Adherence to the File Mapping: conventions guarantees that conforming software will be able to find and read the data.

Adherence to the Data Definitions: guarantees uniformity of meaning among users and between applications.

This part of ISO 10303 generally avoids the storage of redundant data. Sometimes an application may require an alternate (but intellectually equivalent) form of the data; in such cases it is recommended that the alternate form be prepared at the time of use and kept separate from the Fluid Dynamics AP data file. This avoids habitual reliance on the alternate form, which would invalidate the standard. If the alternate form is appended to the file, care must be taken to update the primary (Fluid Dynamics AP) form whenever permanent changes are made.

F.3 Application activity model definitions

The following terms are used in the application activity model. Terms marked with an asterisk are outside the scope of this application protocol. The definitions given in this annex do not supersede the definitions given in the main body of the text.

F.3.1
Acquire and Modify Geometry
The activity of getting the geometric shape of a product and recasting it in a form suitable for an analysis.

F.3.2
Adaptive Modification to FD Model and Process
The activity of revising the computational model and process.

F.3.3
Adjusted Geometry
Modified geometry that is potentially closer to meeting the requirements.

F.3.4
Adjusted Geometry & Modeling Parameters
Adjusted geometry and modeling parameters revised as a result of a prior analysis.

F.3.5
Analysis data
Data resulting from an analysis.

F.3.6
Analysis Objectives
The desired objectives to be met by an analysis.

F.3.7
Approved Data to Customer

Analysis results intended for transmission to a customer. The results have a formal stamp of approval.

F.3.8
Approved Optimized Geometry
Modified geometry that best meets the requirements. The geometry has a formal stamp of approval.

F.3.9
Approved Results
Results from an analysis that have a formal stamp of approval.

F.3.10
Build and Run Analysis Model
The activity of creating a and running a simulation of a product under particular conditions.

F.3.11
Build FD Computational Model
The activity of creating a simulation of a FD process.

F.3.12
Build Product*
The activity of physically making a product.

F.3.13
Build Prototype*
The activity of physically making a prototype of an intended product.

F.3.14
Compute Flowfield
The activity of computing a FD flowfield.

F.3.15
Computing Implementation
One or more particular computer programs.

F.3.16
Concept
The initial ideas about how a product may be realised.

F.3.17
Conduct Analysis of Design
The activity of simulating the performance of a design of a potential product.

F.3.18
Convergence Error

TBD

F.3.19
Definition of Expected Flow Physics
The expected flow process physics to be simulated.

F.3.20
Definition of Expected Physics
The expected physical aspects to be simulated.

F.3.21
Design, Build, Test and Ship Product
The activity encompassing the the lifecycle stages from designing through delivering a product.

F.3.22
Design Product
The activity of designing a product and generating the information required for it to be made.

F.3.23
Develop Product Design*
The activity of designing a product.

F.3.24
Develop Manufacturing Information*
The activity of generating the information required for a designed product to be made.

F.3.25
Engineering data
TBD

F.3.26
Environment
The product's operating environment which should be simulated.

F.3.27
Error estimates
Estimates of errors in an analysis.

F.3.28
Extent & Environment
For a CFD analysis, the overall physical space to be simulated, and the environment.

F.3.29
Extract FD Engineering Data
The activity of generating engineering-related quantities from FD analysis flowfield data.

F.3.30
Extract Results
The activity of generating engineering-related quantities from analysis result data.

F.3.31
Flow Conditions of Interest
The kind of flow conditions to be analysed.

F.3.32
Flowfield data
Detailed numerical data describing a simulated FD flow.

F.3.33
Geometry
Geometry representing the shape of a product. It may be the result of a design (e.g., a blueprint) or from an analysis.

F.3.34
Grid and Boundary Conditions
The mesh and the boundary conditions used for computing a flowfield.

F.3.35
Manufacturing information*
The information necessary to describe how a product should be built.

F.3.36
Materials*
Physical materials required for building a product.

F.3.37
Mathematical Model
The mathematics forming the basis of a simulation.

F.3.38
Mathematical Model of Fluid Dynamics
The mathematics forming the basis of a FD simulation.

F.3.39
Modeling Parameters
TBD

F.3.40
Practices*
Legal, industrial and company specific practices.

F.3.41

Product*
The physical manifestation of a product.

F.3.42
Prototype*
A potential product or physical model of one.

F.3.43
Optimize Geometry*
The activity of modifying geometry to better meet the requirements.

F.3.44
Optimized Geometry

F.3.45
Resolution Equations
TBD

F.3.46
Requirements
The desired product functionality.

F.3.47
Ship Product*
The activity of shipping a product from its place of manufacture.

F.3.48
Staff & Tools*
Personnel and facilities.

F.3.49
Surface Geometry
The geometry representing the surface of a solid.

F.3.50
Test data*
Data resulting from physical tests.

F.3.51
Test Product*
The activity of physically testing a product.

F.3.52
Test Prototype*
The activity of physically testing a prototype.

F.4 Application Process Description

The application activity model diagrams are given in Figures F.1 through F.6. The graphical form of the application activity model is presented in the IDEF0 activity modeling format. Activities and data flows that are out of scope are shown with dashed lines.

Figures F.1 and F.2 show the general very high level activities related to the design, manufacture, and delivery of a product. Analysis is not apparent at this level.

Figure F.3 shows in more detail the general activities related to the design of a product, one of which is analysing potential designs to estimate their performance against the requirements.

Figure F.4 provides more detail on the activities related to the analysis of a design. At this level, particular analysis processes are not specified. The CFD process is particularised at the next lower level.

The first part of the Computational Fluid Dynamics Process is illustrated in Figure F.5 and consists of a number of process components. At a minimum these include:

A1211: Acquire and Modify Geometry;

A1212: Build FD Computational Model;

A1213: Compute Flowfield.

The second, and final, part of the process is illustrated in Figure F.6. This consists of at least:

A1221: Extract FD Engineering Data;

and may further include:

A1222: Adaptive Modification to FD Model and Process, and/or

A1223: Optimize Geometry.

The Fluid Dynamics AP data standard will be utilized throughout this process, as illustrated in Figure F.7.

The initial source of geometry is a Computer-Aided Design (CAD) system, or a prior analysis which could have been a Finite Element or a CFD analysis, or another source. In many cases, it is necessary to edit the geometry that is provided. This editing may be done to remove complexity from the geometry, and thus reduce the cost and cycle time of the subsequent CFD analysis. In some cases, the geometry may be modified to represent intentionally a model which is different from the baseline model that was represented in the original geometry definition. These modifications, if required, represent the first stage of the analysis process: 'Acquire and Modify Geometry' (Process A1211). The product of this process stage is a geometry file representing the surfaces over which fluid may flow.

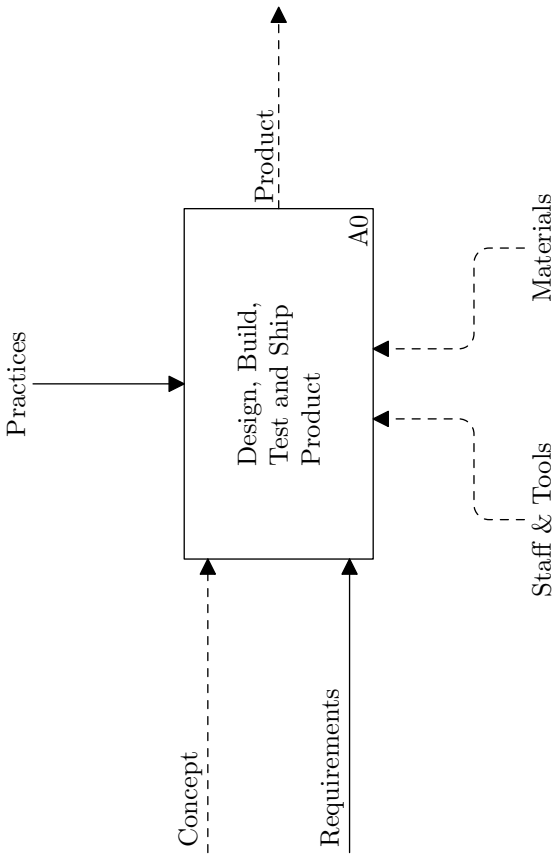


Figure F.1 – A0 Design, build, test and ship product

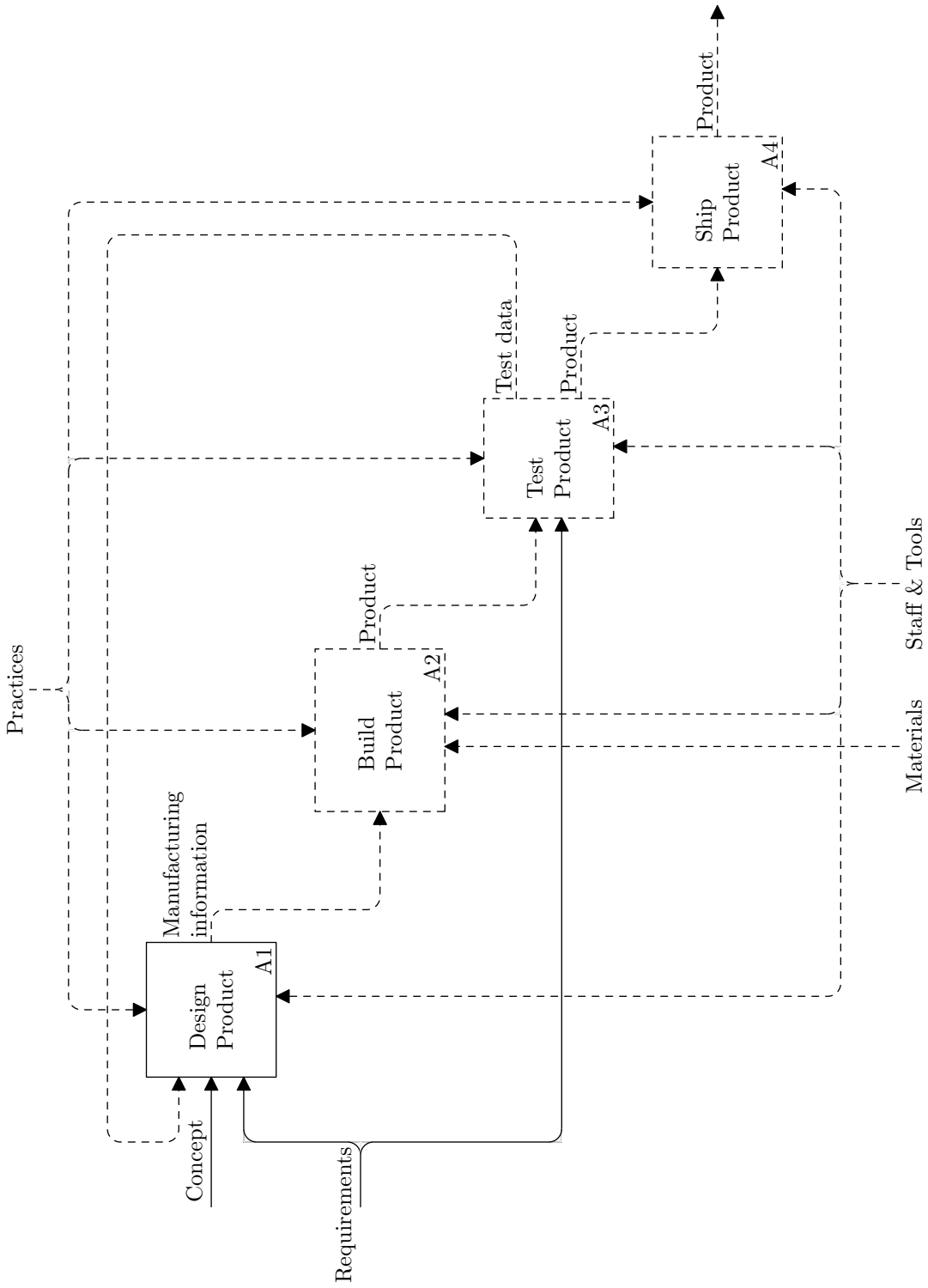


Figure F.2 – A0 Design, build, test and ship product

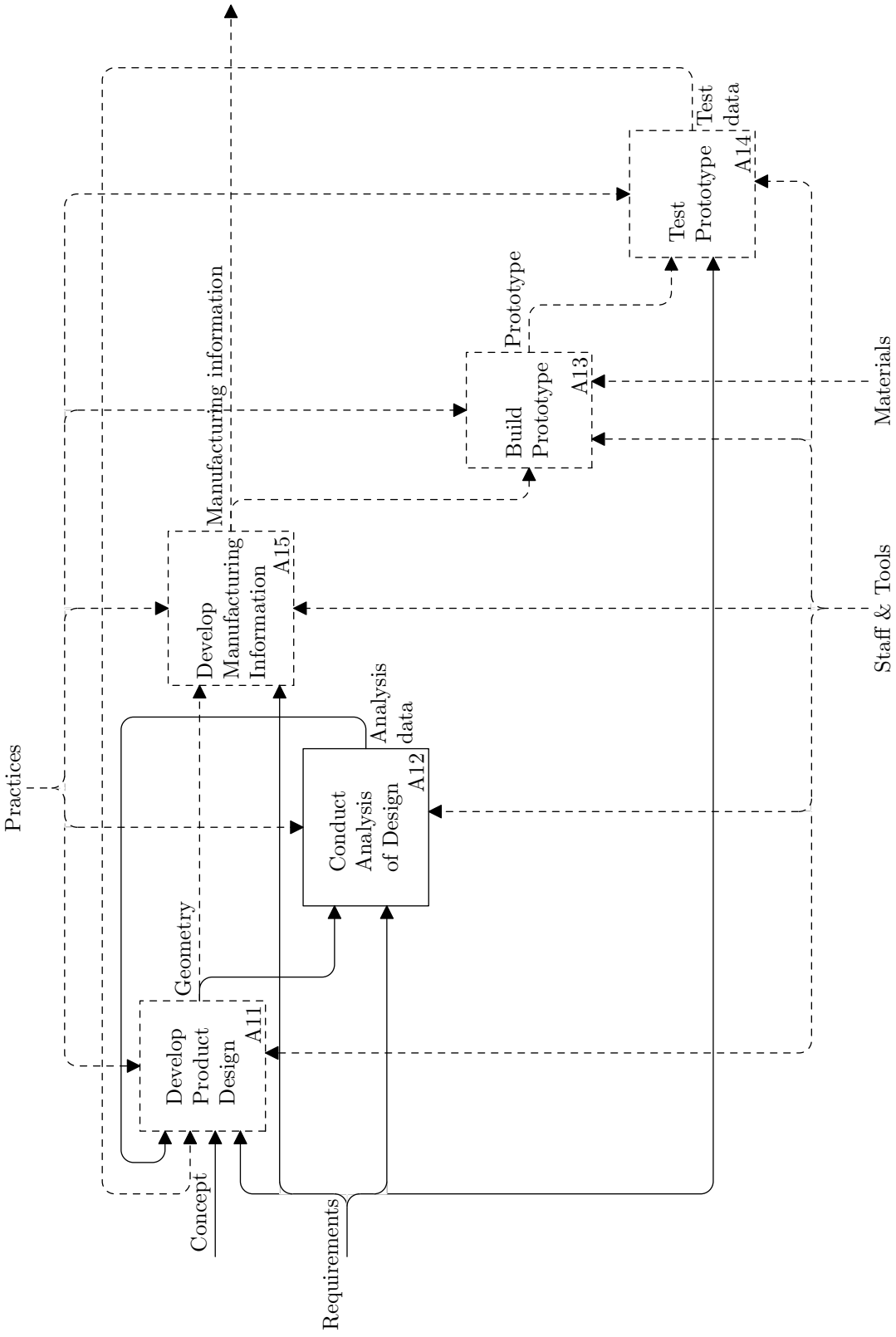


Figure F.3 – A1 Design product

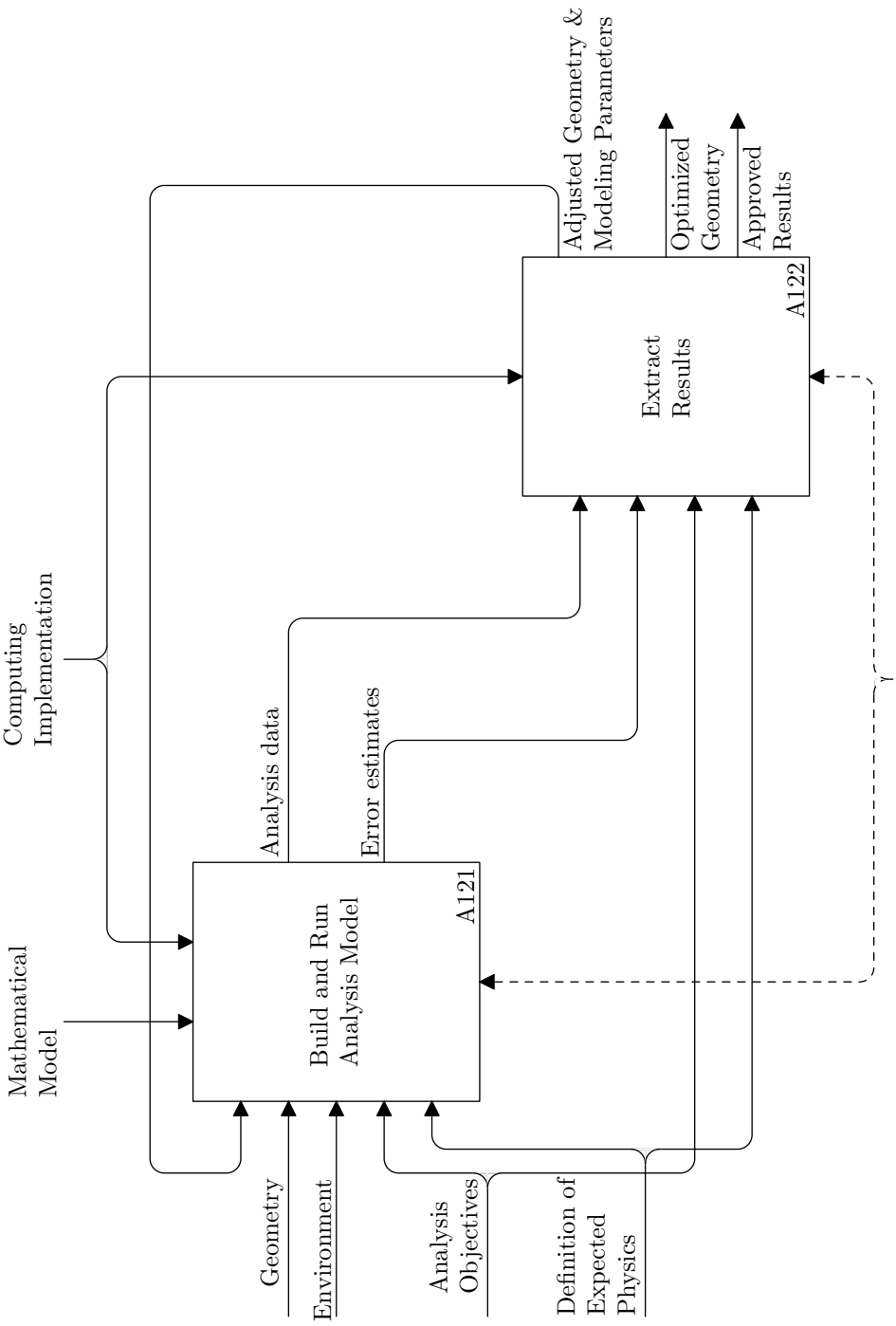


Figure F.4 – A12 Conduct analysis of design

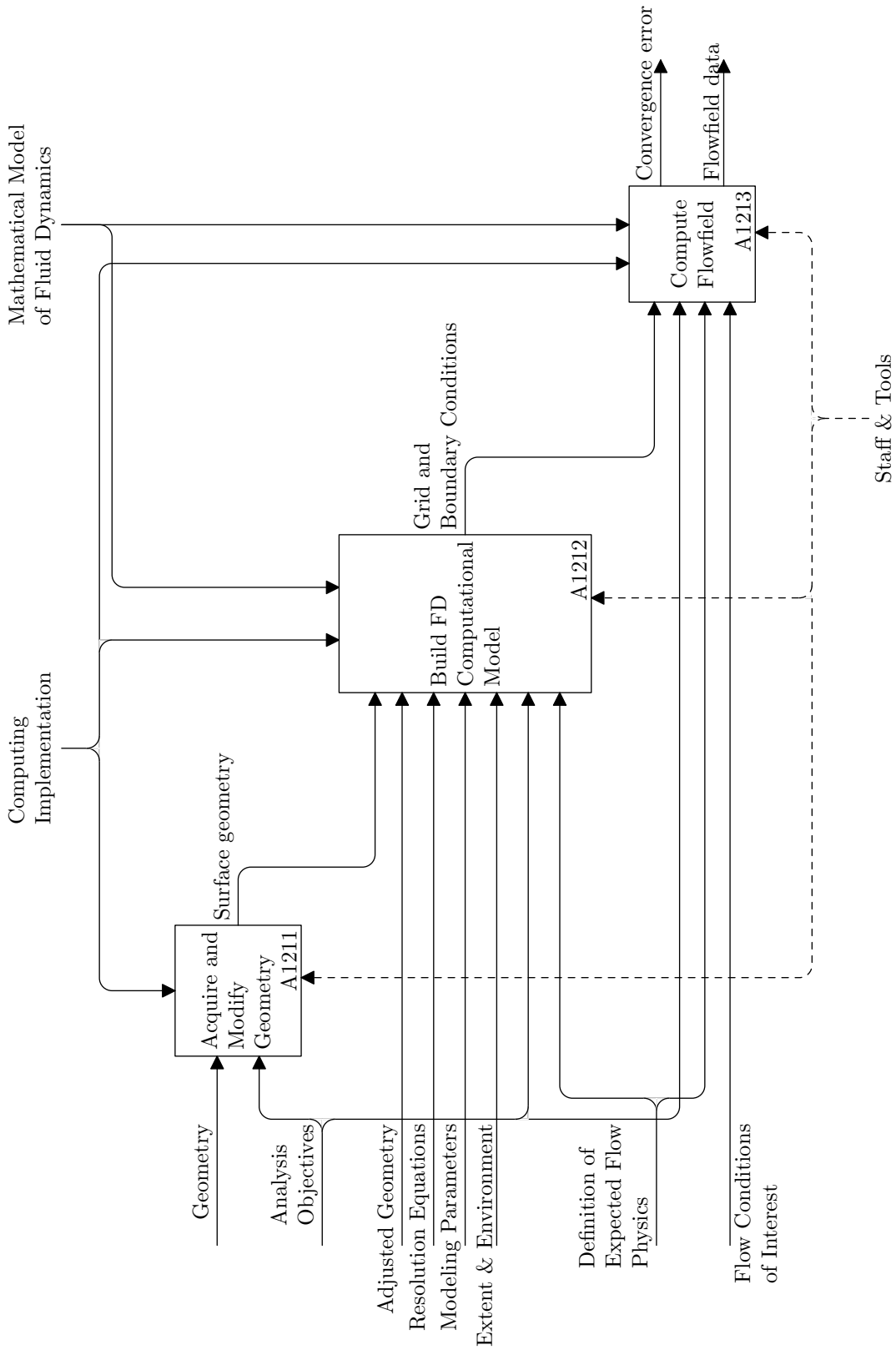


Figure F.5 – A121 Build and run analysis model

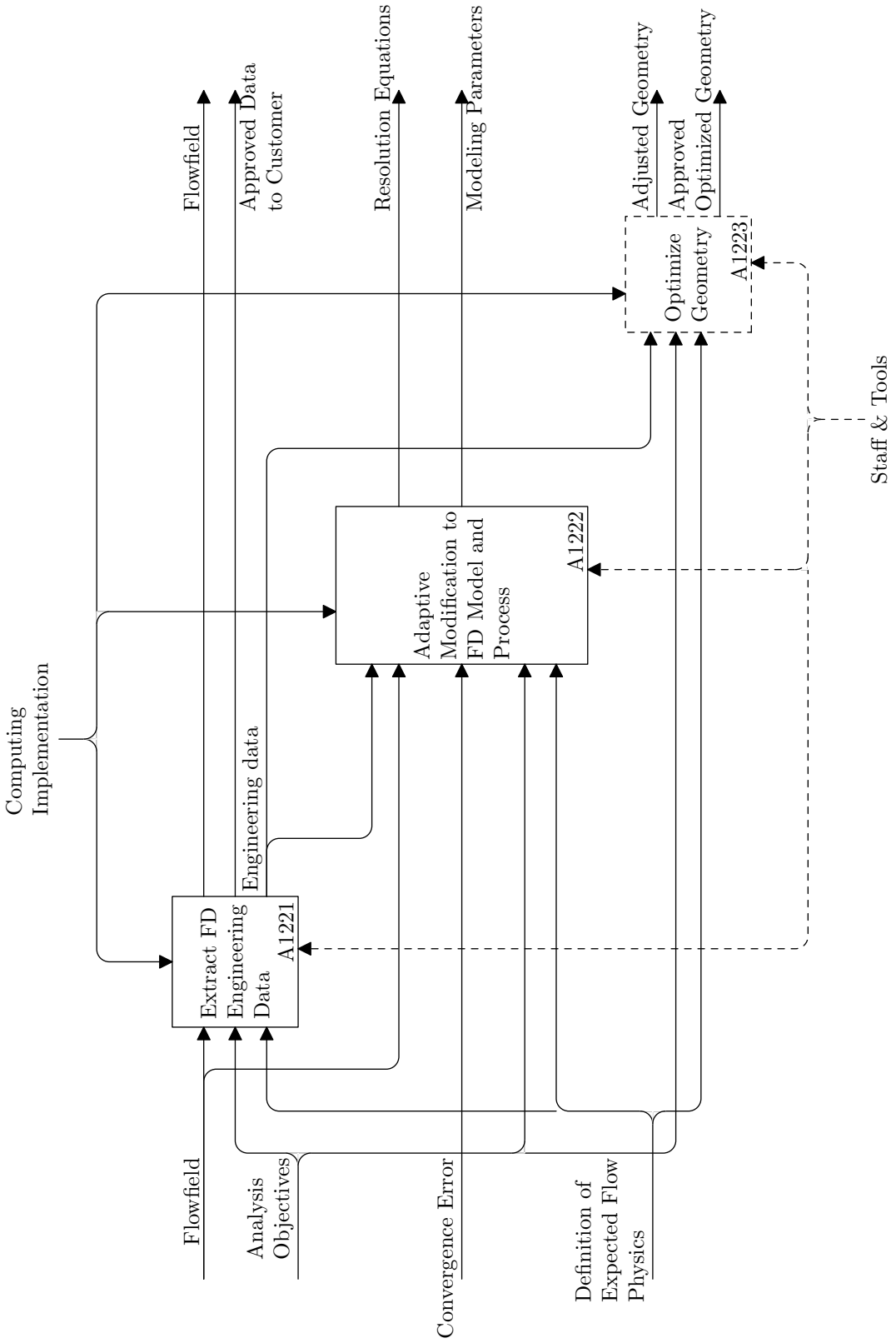


Figure F.6 – A122 Extract results

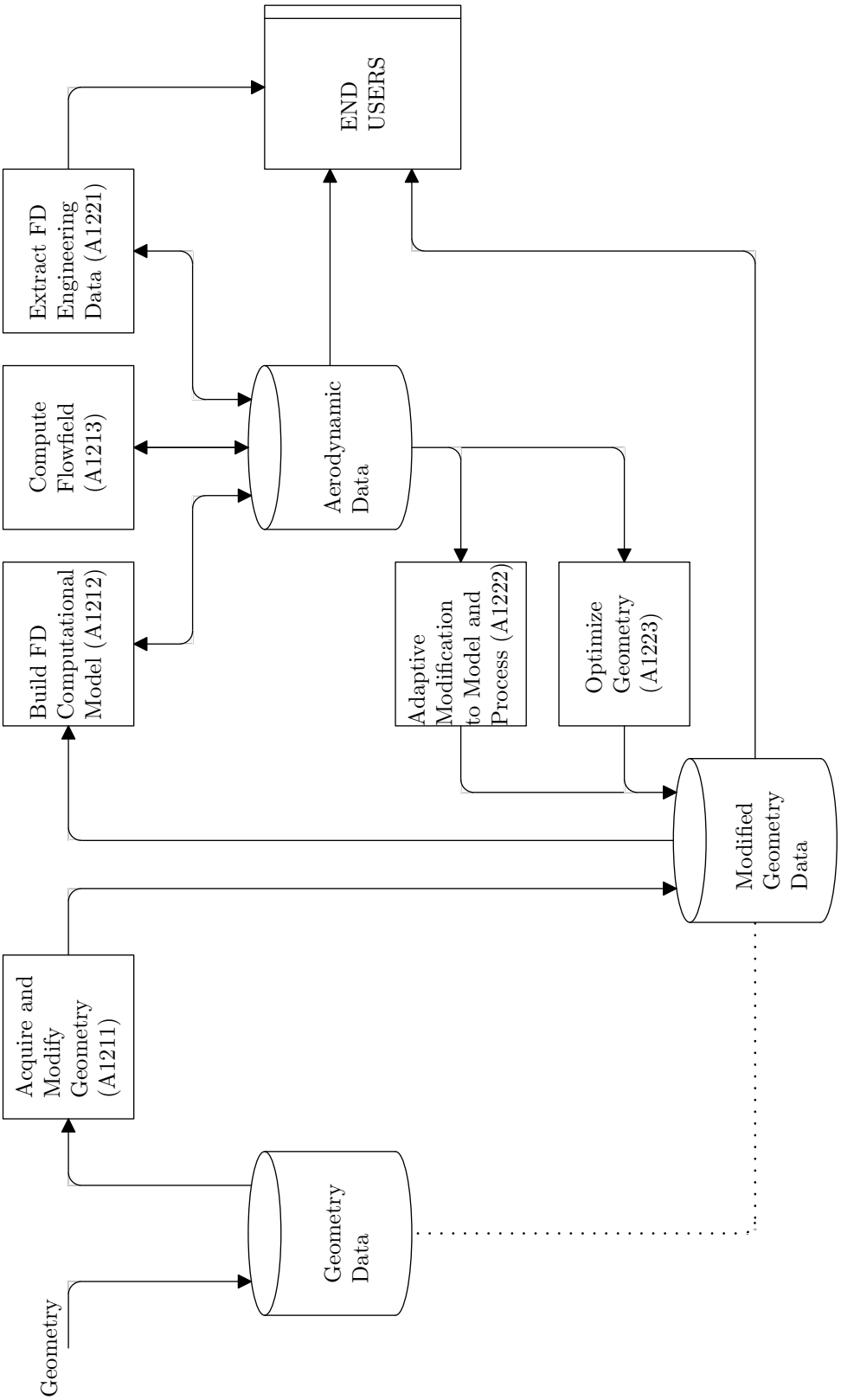


Figure F.7 – Data flow through the CFD process

The CFD analysis process consists of three major stages:

a) Build FD Computational Model (Process A1212) — Using the provided geometry, build the mathematical model and required coordinate systems for the subsequent CFD analysis. The product of this stage is the computational model, represented in the Fluid Dynamics AP format.

b) Compute Flowfield (Process A1213) — Apply the CFD analysis program(s) to the mathematical model to produce the simulation of the aerodynamic flowfield. The flowfield predictions are captured in the Fluid Dynamics AP format.

c) Extract FD Engineering Data (Process A1221) — Extract from the complete flowfield simulation, the data required to meet the end-user's objectives. Typically, these are reduced data such as the net forces which are applied to components of the geometry through the action of the aerodynamic flowfield. The products of this stage of the process may be user-ready data as graphs, printed data, reports, etc., or the products may be additional information which is captured in the Fluid Dynamics AP format.

In some instances, the process also may include one or two recursive loops:

a) Adaptive Modification to FD Model and Process (Process A1222) — In this loop, the characteristics of the flow solver (Process A1213) or the computational model (Process A1212) are modified to improve accuracy or reduce cost of the predictions. These adjustments are controlled by an adaptive algorithm, and by intermediate characteristics of the predicted flowfield. The products of this stage are modifications to the mathematical model and the parameters of the flowfield simulation, stored in the Fluid Dynamics AP format.

b) Optimize Geometry (Process A1223) — In this recursive loop, the analysis geometry is modified based on intermediate flowfield predictions. This geometry modification is made to optimize some attribute of the interaction between the geometry and the flowfield. The products of this stage of the process are: modified geometry; and/or a modified mathematical model or flowfield simulation (Fluid Dynamics AP format).

F.5 Process Variant Approaches

The CFD analysis process includes a number of variant approaches. The relationship of these variant approaches to this part of ISO 10303 is discussed in this subclause.

F.5.1 Equations of Fluid Dynamics

Many different sets of equations of fluid dynamics are used as the basis of a CFD process. Some of the more common mathematical models are known as:

— Navier-Stokes Equations

— Euler Equations

— Nonlinear Potential Flow Equation

— Linear Potential Flow Equation

— Small-Disturbance Equations

— Boundary Layer Equations

— Stream Function Equations

Each of these top-level mathematical models has a number of sub-models, in areas such as computational grid (see F.5.2), flow physics models (see F.5.3), discretization, boundary and initial conditions, and solution algorithm. The CFD analysis process flow is roughly the same for all of these approaches.

F.5.2 Computational Grid

The computational grid provides a local coordinate system for the solution of the mathematical model governing the fluid dynamics problem at issue. Often, the coordinate systems are quite complex, as they as transformed and warped to conform to the details of the geometry to be analyzed. For complex geometries, it is common to introduce multiple independent coordinate systems in subdomains of the flowfield — these subdomains often are called 'blocks'. Together, all the blocks or subdomains must span the entire flowfield that is to be analyzed.

Several broad types of computational grids are embraced within this part of ISO 10303. These types include:

— Structured grid.

— Structured multi-block 1:1 abutting grid (the coordinate systems are continuous across the boundaries between adjacent blocks).

— Structured multi-block mismatched abutting grid (the coordinate systems are discontinuous across the boundaries between adjacent blocks).

— Structured overset grids (the coordinate systems of adjacent blocks will overlap to a material degree).

— Unstructured tetrahedral grids

— Unstructured prismatic grids

— Unstructured arbitrary N-sided grids.

The three types of multi-block structured grids are illustrated in Figure G.14 through Figure G.16.

F.5.3 Flow Physics Models

The equations of fluid dynamics (see F.5.1) describe the conservation of mass, momentum, and energy in a moving fluid. These equations are not adequate, by themselves, to carry out predictions. Additional equations are needed to describe the relationship between thermodynamics properties of the fluid (pressure, temperature, density, viscosity, thermal conductivity, etc). Depending on the requirements of a specific analysis, additional sets of mathematical models may be required to describe the overall impact of turbulence, to describe the impact of chemical reactions such as combustion, and to describe the interaction of an electrically conducting fluid in the presence of electromagnetic fields.

Annex G
(informative)
Application reference model

G.1 Introduction

This annex provides the application reference model for this part of ISO 10303. The application reference model is a representation of the structure and constraints of the application objects specified in clause 4. The application reference model is presented in both EXPRESS and EXPRESS-G. The application reference model is independent from any implementation method. EXPRESS-G is defined in annex D of ISO 10303-11.

The major goal of this ARM is a comprehensive and unambiguous description of the intellectual content of information that must be passed from code to code in a structured-grid multiblock Navier-Stokes analysis system. This information includes grids, flow solutions, multiblock interface connectivity, boundary-conditions, reference states, and dimensional units or normalization associated with data.

The goal is the description of data sets typical of CFD analysis, which tend to contain a small number of extremely large data arrays.

The ARM describes a hierarchical database, precisely defining both the data and their hierarchical relationships.

There are two major alternatives to organizing a CFD hierarchy: topologically based and data-type based. In a topologically based graph, overall organization is by multiblock zones; information pertaining to a particular zone, including its grid coordinates or flow solution, hangs off the zone. In a data-type based graph, organization is by related data. For example, there would be two nodes at the same level, one for grid coordinates and another for the flow solution. Hanging off each of these nodes would be seperate lists of the zones.

The hierarchy described here is topologically based; a simplified illustration of the database hierarchy is shown in Figure G.1. Hanging off the root 'node' of the database is a node containing global reference-state information, such as freestream conditions, and a list of nodes for each zone. The figure shows the nodes that hang off the first zone; similar nodes would hang off of each zone in the database. Nodes containing the physical-coordinate data arrays (x, y and z) for the first zone are shown hanging off the 'grid coordinates' node. Likewise, nodes containing the first zone's flow-solution data arrays hang off the 'flow solution' node. The figure also depicts nodes containing multiblock interface connectivity and boundary condition information for the first zone; subnodes hanging off each of these are not pictured.

Conceptually, an analysis can be thought of as having four components, as in this extremely brief model:

```
ENTITY analysis;
  domain     : computational_space;
```

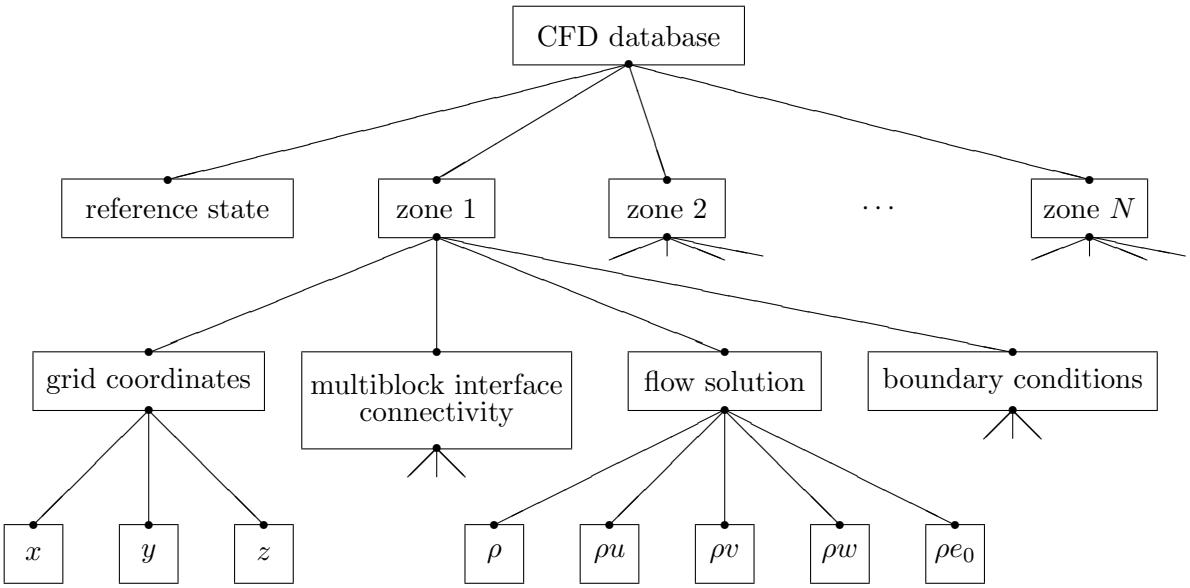



Figure G.1 – Topologically based CFD hierarchy

```
  equations  : SET OF equation;
  conditions : SET OF condition;
  results    : SET OF solution;
END_ENTITY;
```

The **domain** is the representation of the computational space (typically a discrete approximation to a continuum). The **equations** are the mathematical formulations of the physics of the problem at hand and the **conditions** are the conditions (e.g., boundary conditions or constraints) that apply to the particular analysis. Finally, the **results** are the calculated solutions to the equations subject to the conditions over the domain, and possibly other data resulting from the analysis.

The ARM model was initially based on *CGNS Standard Interface Data Structures* [1], together with extensions proposed in *SIDS additions/modifications to support unstructured meshes and geometry links* [2]. These two documents are now merged into a new version *The CFD General Notation System: Standard Interface Data Structures* [3], and this is the basis for the current ARM model.

The model consists of a set of schemas and the relationships between these are shown in Figure G.2.

G.2 hierarchy

The following EXPRESS declaration begins the **hierarchy** schema and identifies the necessary external references.

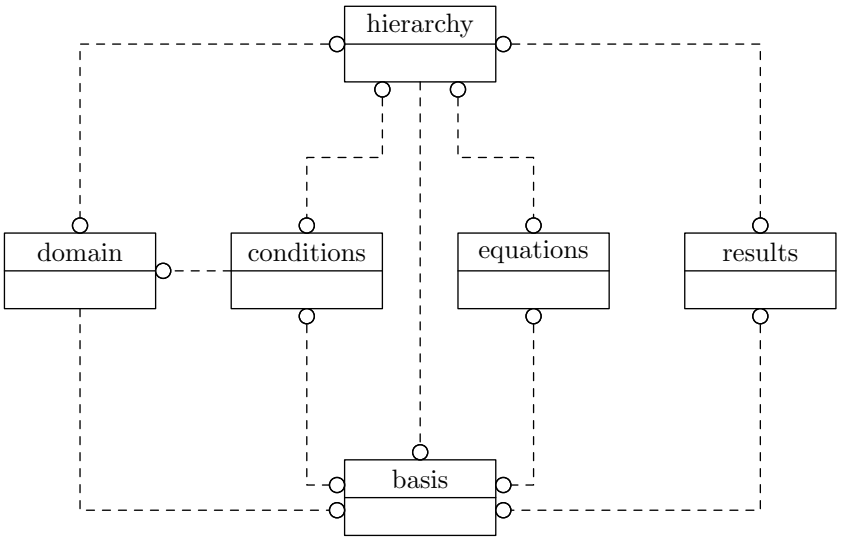


Figure G.2 – Schema level ARM diagram (page 1 of 1)

EXPRESS specification:

```
*)
{iso standard 10303 part (11) version (4)}
SCHEMA hierarchy;
  REFERENCE FROM basis;
  REFERENCE FROM domain;
  REFERENCE FROM conditions;
  REFERENCE FROM equations;
  REFERENCE FROM results;
(*
```

G.2.1 Introduction

This schema defines and describes the structure types for the top levels of the CFD hierarchy. The hierachy is topologically based, where the overall organization is by multiblock zones.

The graphical form for the **hierarchy** schema is given in Figures G.3 through G.6.

G.2.2 Fundamental concepts and assumptions

All information pertaining to a given zone, including grid coordinates, flow solution, and other related data, is contained within that zone's structure entity.

Structures for describing or annotating the database are provided; these same descriptive mechanisms are provided at all levels of the hierarchy.

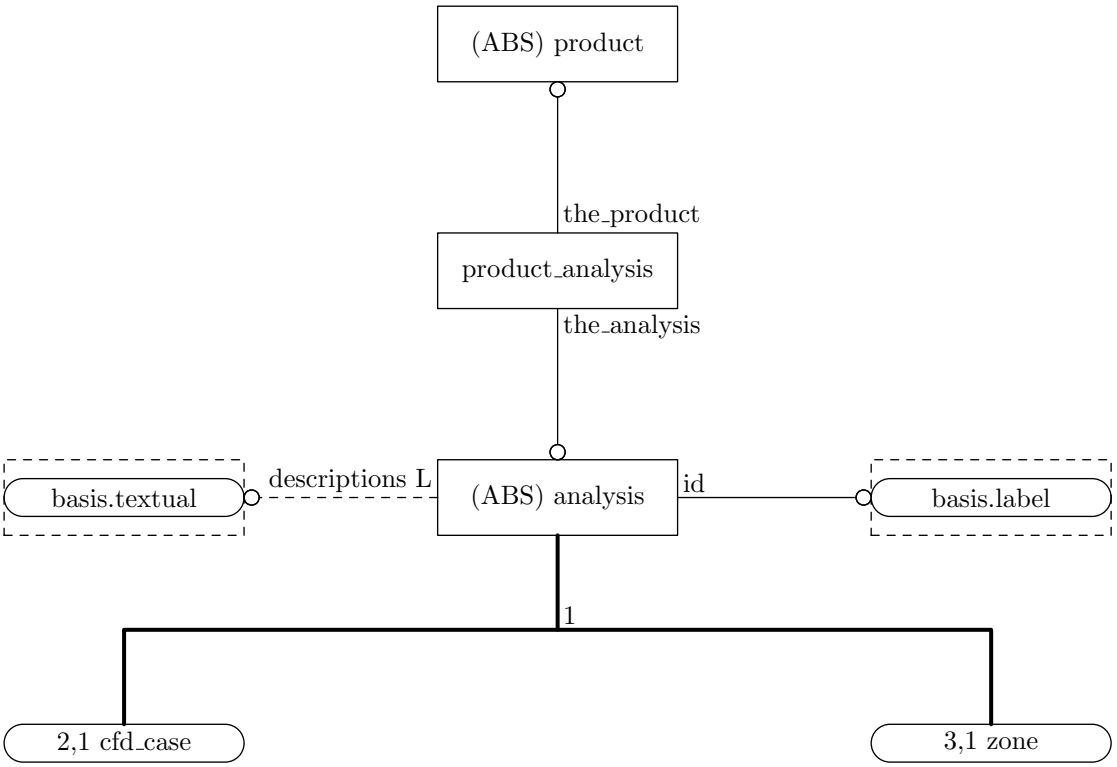


Figure G.3 – Entity level diagram of ARM hierarchy schema (page 1 of 4)

G.2.3 hierarchy entity definitions

G.2.3.1 analysis

An **analysis** represents the concept of a mathematical and/or numerical anlaysis.

EXPRESS specification:

```
*)
ENTITY analysis;
  descriptions       : OPTIONAL LIST OF textual;
  id                 : label;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_analysis FOR analysis;
  ABSTRACT SUPERTYPE;
  ONEOF(cfd_case,
        zone);
END_SUBTYPE_CONSTRAINT;
(*
```

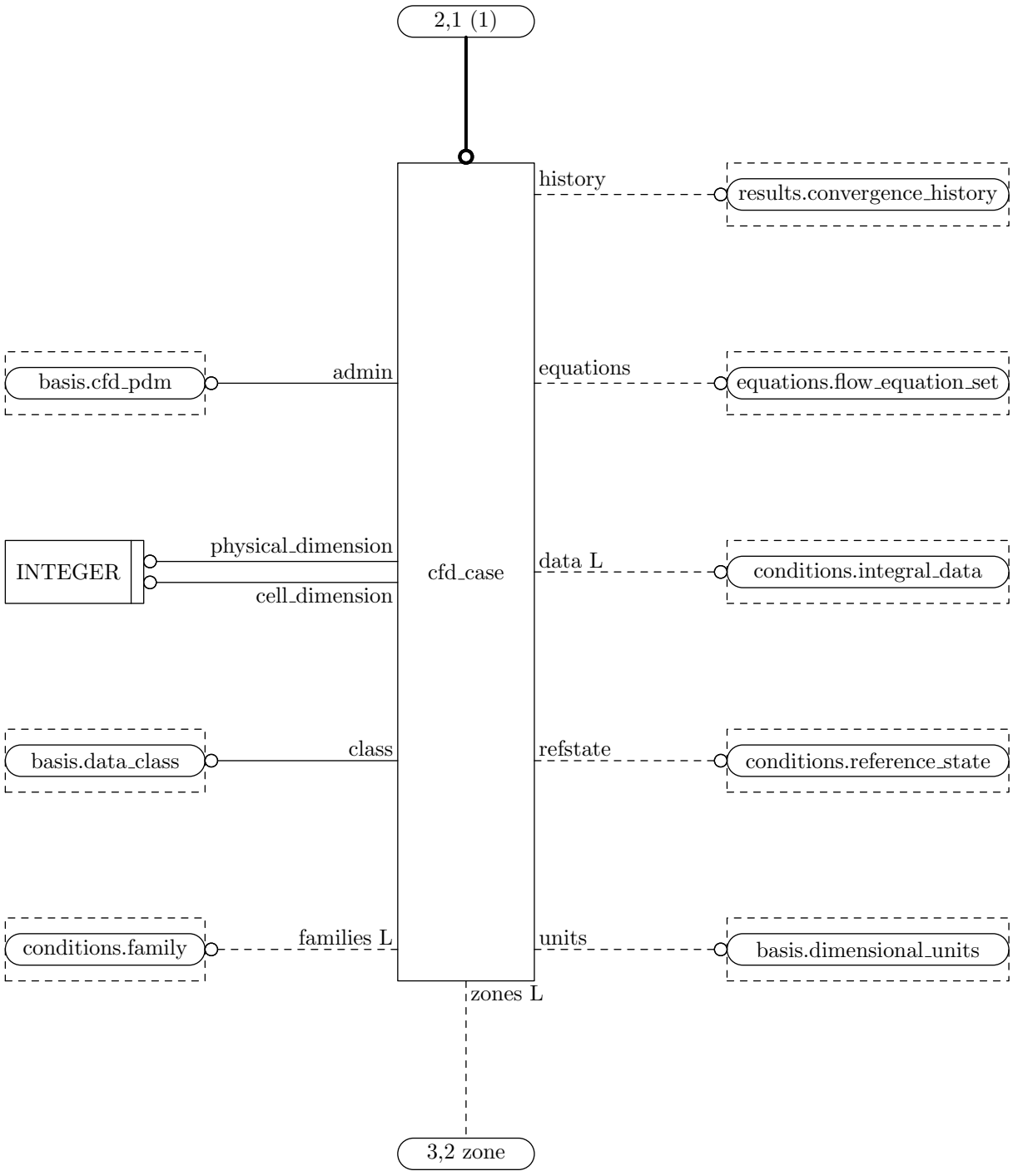



Figure G.4 – Entity level diagram of ARM hierarchy schema (page 2 of 4)

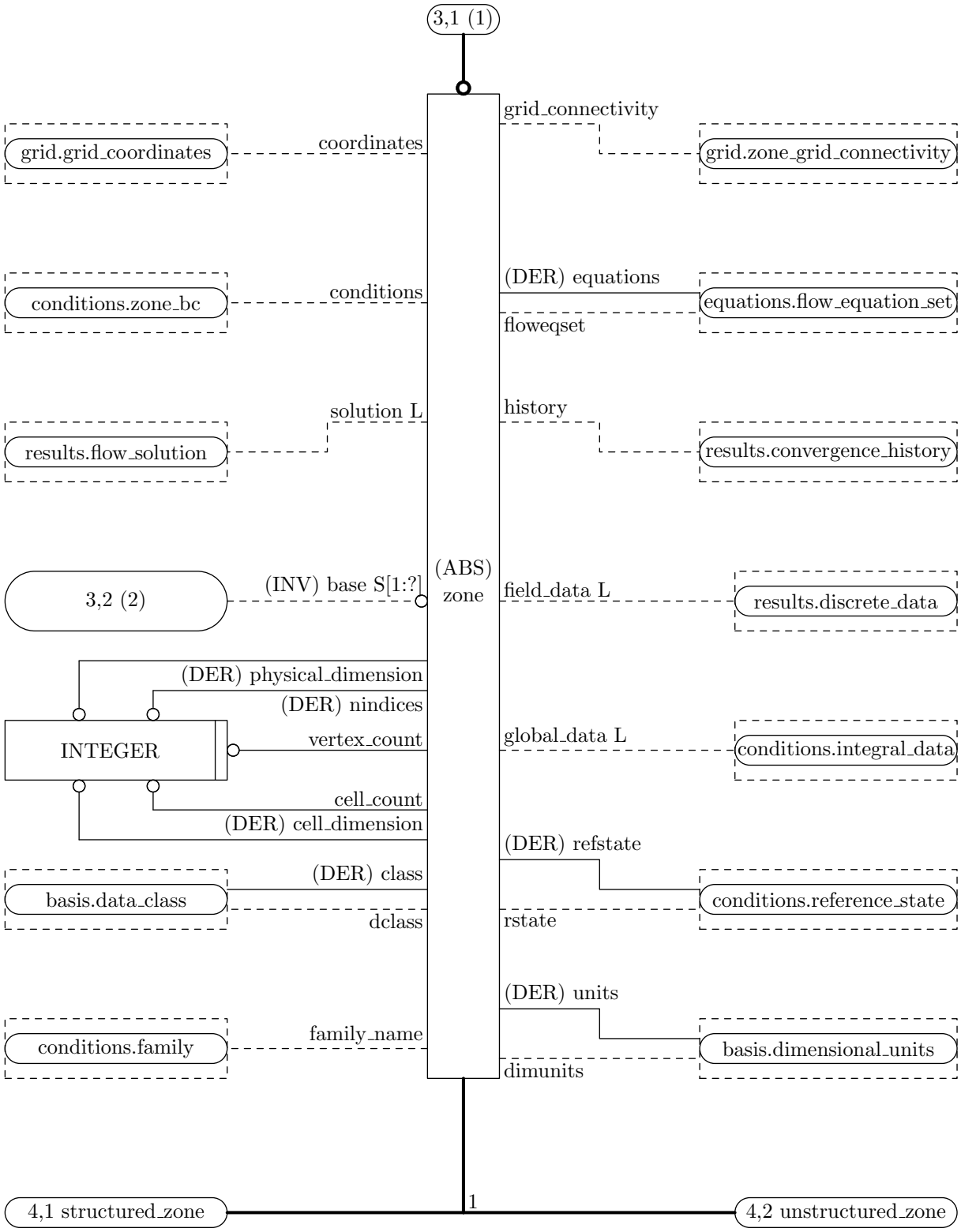


Figure G.5 – Entity level diagram of ARM hierarchy schema (page 3 of 4)

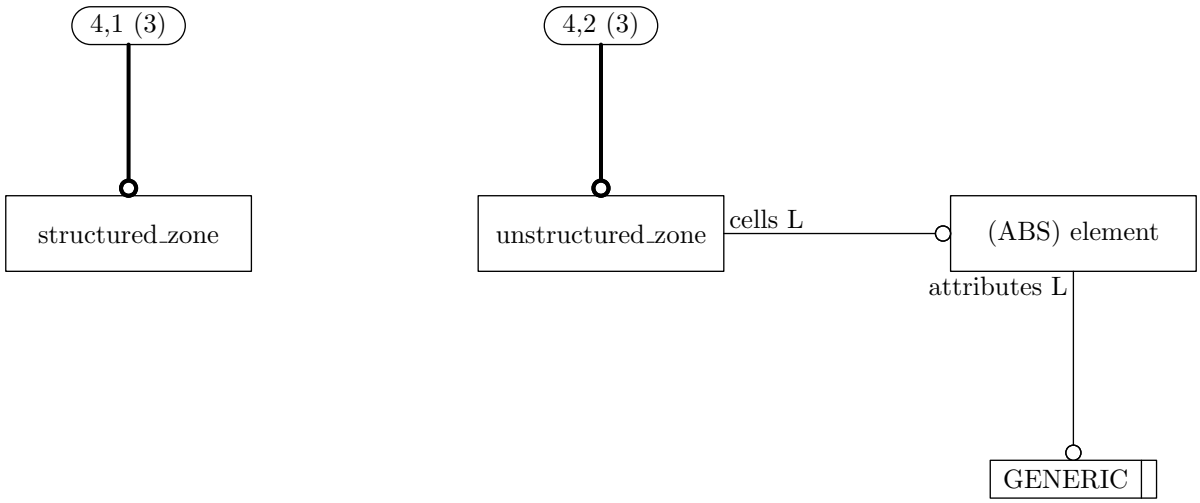


Figure G.6 – Entity level diagram of ARM hierarchy schema (page 4 of 4)

Attribute definitions:

descriptions: Annotations;

id: User-specified instance identifier;

G.2.3.2 cfd_case

The highest level structure in a CFD database is **cfd_case**. It contains the dimensionality of the grid and a list of zones making up the domain. Globally applicable information, including a reference state, a set of flow equations, dimensional units, and convergence history are also attached. In addition, structures for describing or annotating the database are also accomodated.

EXPRESS specification:

```
*)
ENTITY cfd_case
  SUBTYPE OF (analysis);
  cell_dimension     : INTEGER;
  physical_dimension : INTEGER;
  zones              : OPTIONAL LIST OF zone;
  refstate           : OPTIONAL reference_state;
  class              : data_class;
  units              : OPTIONAL dimensional_units;
  equations          : OPTIONAL flow_equation_set;
  history            : OPTIONAL convergence_history;
  data               : OPTIONAL LIST OF integral_data;
  families           : OPTIONAL LIST OF family;
  admin              : cfd_pdm;
```



```
END_ENTITY;
(*
```

Attribute definitions:

cell_dimension: The dimension of cells in the mesh.

physical_dimension: The number of coordinates required to define a node position.

zones: Data specific to each zone or block in a multiblock case. The size of the list defines the number of zones or blocks in the domain.

refstate: Reference data applicable to the entire database; quantities such as Reynolds number and freestream Mach number are given here (for external flow problems).

class: Global default data class for the database. If the CFD database contains dimensional data (e.g., velocity with units of m/s), **units** may be used to describe the system of units employed.

units: Specification of the global default units;

equations: Description of the governing flow equations associated with the entire database. This structure contains information on the general class of governing equations (e.g., Euler or Navier-Stokes), equation sets required for closure, including turbulence modelling and equations of state, and constants associated with the equations.

history: Global relevant convergence history. The convergence information includes total configuration forces, global parameters (e.g., freestream angle-of-attack), and global residual and solution-change norms taken over all the zones.

data: Miscellaneous data. Candidates for inclusion are global forces and moments.

families: Global family information;

admin: Administrative and Product Data Management data.

class, **units**, **refstate** and **equations** have special function in the CFD hierarchy. They are globally applicable throughout the database, but their values may be superseded by local entities (e.g., within a given zone).

G.2.3.3 product_analysis

product_analysis captures a relationship between a product and an analysis of (part of) the product.

EXPRESS specification:

```
*)
ENTITY product_analysis;
```



```
  the_product  : product;
  the_analysis : cfd_case;
END_ENTITY;
(*
```

Attribute definitions:

the_product: the product under analysis;

the_analysis: the analysis of **the_product**.

G.2.3.4 product

A product.

NOTE **product** is defined in ISO 10303-41.

EXPRESS specification:

```
*)
ENTITY product
  ABSTRACT;
END_ENTITY;
(*
```

G.2.3.5 zone

zone contains all information pertinent to an individual multiblock zone. This information includes the number of cells and vertices making up the grid, the physical coordinates of the grid vertices, the flow solution, multiblock interface connectivity, boundary-conditions, and zonal convergence-history data. In addition this structure contains a reference state, a set of flow equations and dimensional units that are all unique to the zone.

EXPRESS specification:

```
*)
ENTITY zone
  SUBTYPE OF (analysis);
  vertex_count      : ARRAY [1:nindices] OF INTEGER;
  cell_count        : ARRAY [1:nindices] OF INTEGER;
  coordinates       : OPTIONAL grid_coordinates;
  family_name       : OPTIONAL family;
  solution          : OPTIONAL LIST OF flow_solution;
  field_data        : OPTIONAL LIST OF discrete_data;
  global_data       : OPTIONAL LIST OF integral_data;
```



```
  grid_connectivity : OPTIONAL zone_grid_connectivity;
  conditions        : OPTIONAL zone_bc;
  rstate            : OPTIONAL reference_state;
  dclass            : OPTIONAL data_class;
  dimunits          : OPTIONAL dimensional_units;
  floweqset         : OPTIONAL flow_equation_set;
  history           : OPTIONAL convergence_history;
DERIVE
  cell_dimension     : INTEGER := base.cell_dimension;
  physical_dimension : INTEGER := base.physical_dimension;
  nindices             : INTEGER := derive_zone_dimension(SELF);
  class              : data_class := NVL(dclass, base.class);
  units              : dimensional_units := NVL(dimunits, base.units);
  equations          : flow_equation_set:= NVL(floweqset, base.equations);
  refstate           : reference_state := NVL(rstate, base.refstate);
INVERSE
  base : SET [1:?] OF cfd_case FOR zones;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_zone FOR zone;
  ABSTRACT SUPERTYPE;
  ONEOF(structured_zone,
        unstructured_zone);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

vertex_count: is the number of vertices in each index direction. it is the number of vertices defining 'the grid' or the domain (i.e., without rind points).

cell_count: is the number of cells in each index direction. It is the number of cells on the interior of the domain.

coordinates: are the physical coordinates of the grid vertices. This structure defines 'the grid'; it may optionally contain physical coordinates of rind or ghost points.

family_name: Identifies to which family the zone belongs to. Family names may be used to define material properties.

solution: is the flow-solution quantities. Each instance of **flow_solution** shall only contain data at a single grid location (vertices, cell-centers, etc.); therefore, multiple **flow_solution** structures are provided to store flow-solution data at different grid locations. These structures may optionally contain solution data defined at rind points.

field_data: is miscellaneous field data. Candidate information includes residuals, fluxes and other discrete data that is considered auxiliary to the flow solution.

global_data: is miscellaneous zone-specific global data, other than reference-state data and convergence history information.

grid_connectivity: is the multiblock interface-connectivity information.

conditions: is the boundary-condition information.

rstate: non-default reference-state data.

dclass: non-default class of data.

dimunits: non-default system of units.

floweqset: if a set of flow equations are specific to an individual zone, these are described here.

EXAMPLE 1 For example, if a single zone in the domain is inviscid, whereas all others are turbulent, then this zone-specific equation set could be used to describe the special zone.

history: is the convergence history of the zone; this includes residual and solution-change norms.

cell_dimension: The dimension of a cell in the mesh.

physical_dimension: The number of coordinates required to define a node position.

nindices: The number of indices required to identify uniquely a vertex or a cell in the grid. It is the indexical dimensionality of the computational grid. For structured-grid calculations, **nindices** is usually the same as the spatial problem being solved (e.g., **nindices**=3 for a 3-D problem). For lower-dimensional flowfields, such as quasi 3-D flow, **nindices** may not be the same as the dimensionality of the position vector or the velocity vector. For unstructured grids, usually **nindices**=1 since all the grid points and flow solution variables are stored in 1-D arrays. However, there are instances, such as prismatic boundary-layer grids, where **nindices** may be 2.

class: is the zonal default for the class of data contained in the zone and its substructures.

units: is the description of the system of dimensional units in the zone.

refstate: is reference-state data specific to the individual zone.

equations: is the fow equation set.

base: is the database.

G.2.3.6 structured_zone

structured_zone contains the information pertinent to an individual structured multiblock zone.

EXPRESS specification:

```
*)
ENTITY structured_zone
  SUBTYPE OF (zone);
END_ENTITY;
(*
```


NOTE 1 For structured grids in 3-D, `CellSize = VertexSize - [1,1,1]`.

G.2.3.7 unstructured_zone

unstructured_zone contains the information pertinent to an individual unstructured zone.

EXPRESS specification:

```
*)
ENTITY unstructured_zone
  SUBTYPE OF (zone);
  cells : LIST OF element;
END_ENTITY;
(*
```

Attribute definitions:

cells: the connected elements comprising the zone.

G.2.3.8 element

An **element** is a cell in an unstructured grid. The shape of a cell may be a simple two-dimensional bar or a three-dimensional hexahedron, or another of a range of shapes.

NOTE 1 Elements are described in detail in ISO 10303-5w.

EXPRESS specification:

```
*)
ENTITY element;
  ABSTRACT;
  attributes : LIST OF GENERIC;
END_ENTITY;
(*
```

Argument definitions:

attributes: the defining values.

G.2.4 hierarchy function definitions

G.2.4.1 derive_zone_dimension

derive_zone_dimension takes a **zone** as an argument and returns the value of the indexical dimensionality of the computational grid. This is the number of indices required to identify a vertex.

For a structured zone the dimensionality is the same as the cell dimension. For an unstructured zone the dimensionality is always 1.

EXPRESS specification:

```
*)
FUNCTION derive_zone_dimension(arg : zone) : INTEGER;
  IF ('structured_zone' IN TYPEOF(arg)) THEN
    RETURN(arg.cell_dimension);
  ELSE
    IF ('unstructured_zone' IN TYPEOF(arg)) THEN
      RETURN(1);
    END_IF;
  END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **zone**.

RETURNS: The inexical dimensionality of the computational grid if the zone is either a structured or an unstructured zone, otherwise it returns indefinate.

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of hierarchy
(*
```

G.3 basis

The following EXPRESS declaration begins the **basis** schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA basis;
  REFERENCE FROM conditions (bc_type, Riemann_1D_data_name);
  REFERENCE FROM equations (turbulence_data_name);
  REFERENCE FROM results (flow_solution_data_name, force_moment_data_name);
(*
```

G.3.1 Introduction

This schema defines and describes low-level structures and types that are used in the definition of more complex structures in the hierarchy.

The graphical form for the **basis** schema is given in Figures G.7 through G.11.

G.3.2 Fundamental concepts and assumptions

The structure type **data_array** is a general purpose structure used for holding data arrays and scalars throughout the CFD hierarchy. It is used to describe grid coordinates, flow-solution data, governing flow parameters, boundary-condition data, and other information. For most of these different types of CFD data, a list of standardized identifiers is provided. For example, the standardized identifier **density** is used for data arrays containing static density data.

Five classes of data are addressed with the **data_array** structure type:

a) dimensional data (e.g., velocity in units of m/s);

b) nondimensional data normalized by dimensional reference quantities;

c) nondimensional data with associated nondimensional reference quantities;

d) nondimensional parameters (e.g., Reynolds number, pressure coefficient);

e) pure constants (e.g., π, e).

Each of the five classes of data requires different information to describe dimensional units or normalization associated with the data.

Identifiers or names can be attached to **data_array** entities to identify and describe the quantity being stored. To facilitate communication between different application codes, a set of standardized data-name identifiers with fairly precise definitions are provided. For any identifier in this set, the associated data should be unambiguously understood. In essence, this schema supplies standardized terminology for labeling CFD-related data, including grid coordinates, flow solution, turbulence model quantities, nondimensional governing parameters, boundary-condition quantities, and forces and moments.

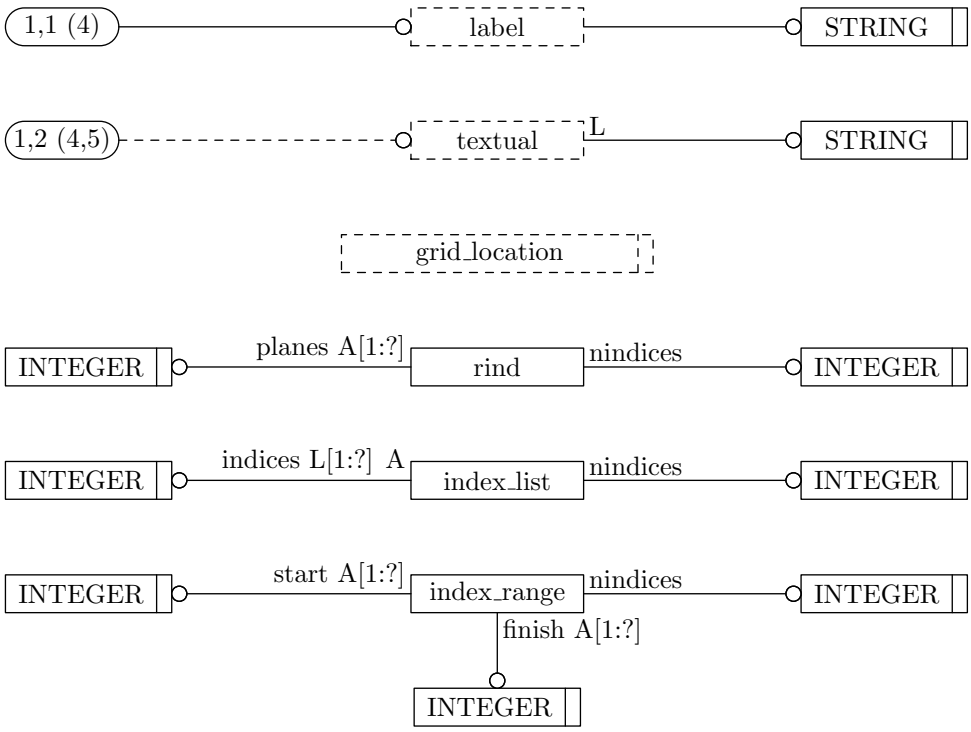


Figure G.7 – Entity level diagram of ARM basis schema (page 1 of 5)

All standardized identifiers denote scalar quantities; this is consistent with the intended use of the **data_array** structure type to describe an array of scalars. For quantities that are vectors, such as velocity, their components are listed.

Included with the lists of standard data-name identifiers, the fundamental units of dimensions associated with that quantity are provided. The following notation is used for the fundamental units: **M** is mass, **L** is length, **T** is time, Θ is temperature and α is angle. These fundamental units are directly associated with the elements of the **dimensional_exponents** structure. For example, a quantity that has dimensions **ML/T** corresponds to **MassExponent** = +1, **LengthExponent** = +1, and **TimeExponent** = -1.

All quantities in the following subclauses denote REAL data types.

G.3.3 basis type definitions

G.3.3.1 label

label is an alphanumeric string which represents the human-interpretable name of something and has a natural language meaning.

NOTE 1 This is the same as **label** in ISO 10303-41.

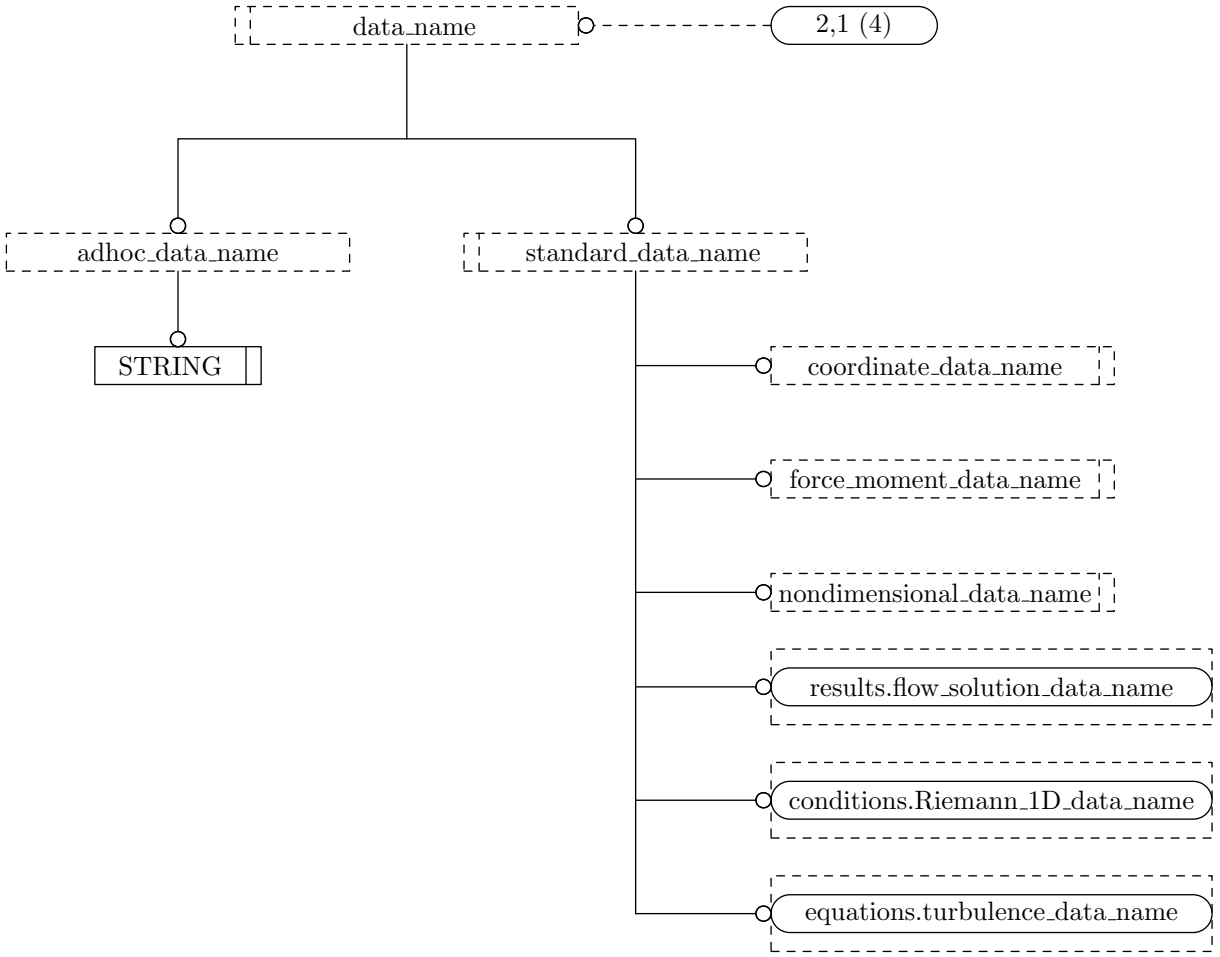


Figure G.8 – Entity level diagram of ARM basis schema (page 2 of 5)

EXPRESS specification:

```
*)
TYPE label = STRING;
END_TYPE;
(*
```

G.3.3.2 textual

textual is a list of alphanumeric strings intended to be read and understood by a human being. It is for information purposes only.

NOTE 1 Effectively, **textual** is a list of **text** as specified in ISO 10303-41.

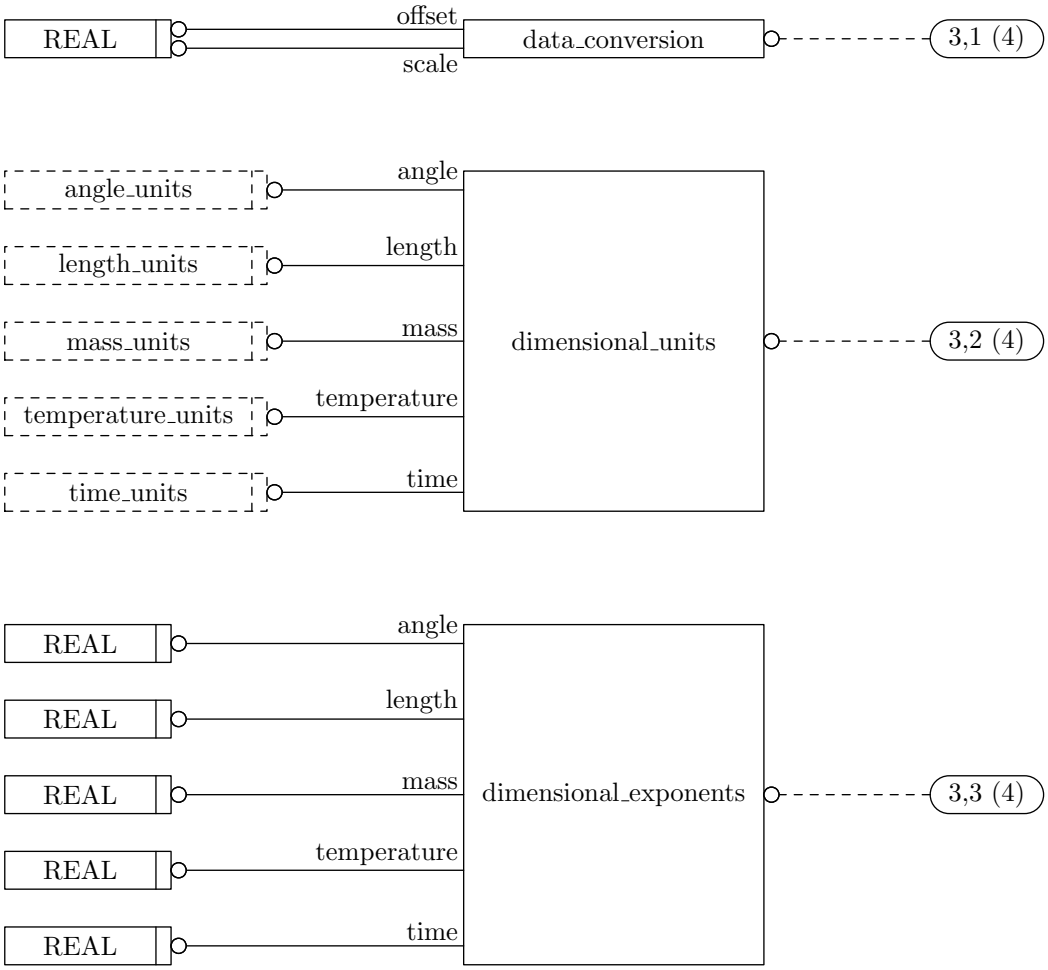


Figure G.9 – Entity level diagram of ARM basis schema (page 3 of 5)

EXPRESS specification:

```
*)
TYPE textual = LIST OF STRING;
END_TYPE;
(*
```

G.3.3.3 data_class

data_class is an enumeration type that identifies the class of a given piece of data.

NOTE 1 Data class is decribed in detail in ISO 10303-5w.

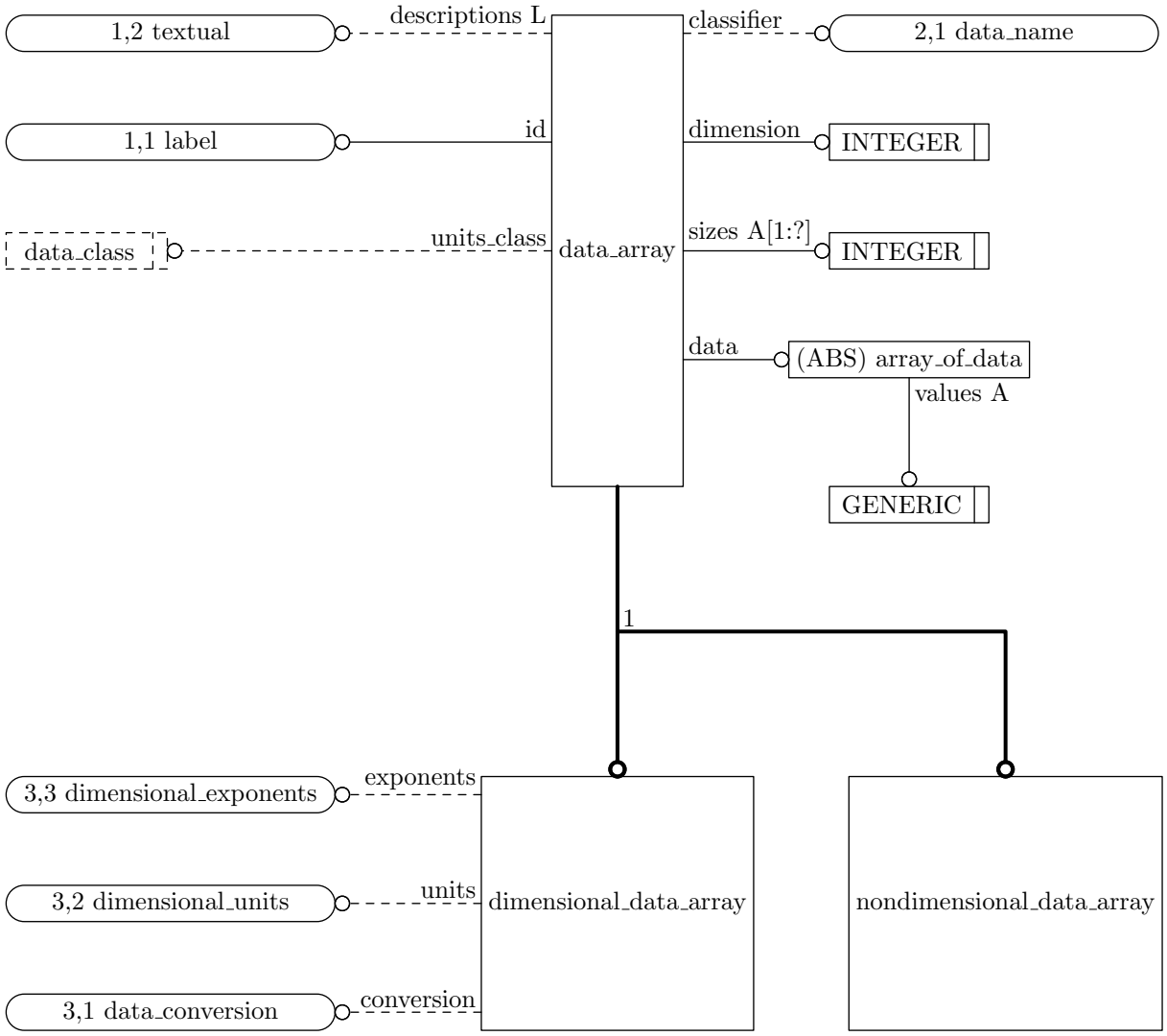


Figure G.10 – Entity level diagram of ARM basis schema (page 4 of 5)

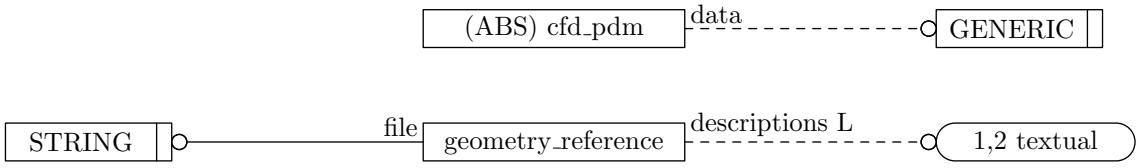


Figure G.11 – Entity level diagram of ARM basis schema (page 5 of 5)

EXPRESS specification:

```
*)
TYPE data_class = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.4 mass_units

An enumeration of units of mass. These include kilogram, gram, slug, and pound mass.

EXPRESS specification:

```
*)
TYPE mass_units = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.5 length_units

An enumeration of units of length. These include meter, centimeter, millimeter, foot, and inch.

EXPRESS specification:

```
*)
TYPE length_units = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.6 time_units

An enumeration of units of time. These include second.

EXPRESS specification:

```
*)
TYPE time_units = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.7 temperature_units

An enumeration of units of temperature. These include Kelvin, Celsius, Rankine, and Fahren-

heit.

EXPRESS specification:

```
*)
TYPE temperature_units = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.8 angle_units

An enumeration of units of plane angle. These include degree and radian.

EXPRESS specification:

```
*)
TYPE angle_units = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.9 grid_location

grid_location is an enumeration of locations with respect to a grid.

NOTE 1 Grid locations are described in detail in ISO 10303-5w.

EXPRESS specification:

```
*)
TYPE grid_location = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.10 data_name

data_name is an identifier for the contents of a **data_array**. It is a superset of **standard_data_name** and **adhoc_data_name**.

EXPRESS specification:

```
*)
TYPE data_name = SELECT
```



```
     (standard_data_name,
      adhoc_data_name);
END_TYPE;
(*
```

G.3.3.11 adhoc_data_name

adhoc_data_name is a STRING providing a non-standard identifier for the contents of a **data_array**.

EXPRESS specification:

```
*)
TYPE adhoc_data_name = STRING;
END_TYPE;
(*
```

G.3.3.12 standard_data_name

standard_data_name is a listing of standardized identifiers for the contents of a **data_array**.

EXPRESS specification:

```
*)
TYPE standard_data_name = EXTENSIBLE SELECT
     (coordinate_data_name,
      nondimensional_data_name,
      Riemann_1D_data_name,
      turbulence_data_name,
      flow_solution_data_name,
      force_moment_data_name);
END_TYPE;
(*
```

G.3.3.13 coordinate_data_name

coordinate_data_name is an enumeration of standardized coordinate systems data. These include cartesian, cylindrical, spherical, and auxiliary.

NOTE 1 Coordinate names are described in detail in ISO 10303-5w.

EXPRESS specification:

```
*)
```



```
TYPE coordinate_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

G.3.3.14 nondimensional_data_name

nondimensional_data_name is an enumeration of standardized nondimensional parameters.

CFD codes are rich in nondimensional governing parameters, such as Mach number and Reynolds number, and nondimensional flowfield coefficients, such as pressure coefficient. The problem with these parameters is that their definitions and conditions that they are evaluated at can vary from code to code. Reynolds number is particularly notorious in this respect.

These parameters have posed us with a difficult dilemma. Either we impose a rigid definition for each and force all database users to abide by it, or we develop some methodology for describing the particular definition that the user is employing. The first limits applicability and flexibility, and the second adds complexity. We have opted for the second approach, but we include only enough information about the definition of each parameter to allow for conversion operations. For example, the Reynolds number includes velocity, length and kinematic viscosity scales in its definition (i.e. $Re = VL/\nu$). The database description of Reynolds number includes these different scales. By providing these 'definition components', any code that reads Reynolds number from the database can transform its value to an appropriate internal definition. These 'definition components' are identified by appending a '_' to the data-name identifier of the parameter.

Definitions for nondimensional flowfield coefficients follow: The pressure coefficient is defined as,

$$c_p = \frac{p - p_{\text{ref}}}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2},$$

where $\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2$ is the dynamic pressure evaluated at some reference condition, and p_{ref} is some reference pressure. The skin friction coefficient is,

$$\vec{c}_f = \frac{\vec{\tau}}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2},$$

where $\vec{\tau}$ is the shear stress or skin friction vector. Usually, $\vec{\tau}$ is evaluated at the wall surface.

EXPRESS specification:

```
*)
TYPE nondimensional_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

The required identifiers and their meanings are given in Table 8.

G.3.4 basis entity definitions

G.3.4.1 cfd_pdm

cfd_pdm is CFD administrative and Product Data Management data; it covers such aspects as dates, responsiblities, status, approvals, etc.

EXPRESS specification:

```
*)
ENTITY cfd_pdm
  ABSTRACT;
  data : GENERIC;
END_ENTITY;
(*
```

Attribute definitions:

data: the administrative and PDM data.

G.3.4.2 array_of_data

array_of_data is a multidimensional array of data values.

EXPRESS specification:

```
*)
ENTITY array_of_data
  ABSTRACT;
  values : ARRAY OF GENERIC;
END_ENTITY;
(*
```

Attribute definitions:

values: the data values. These will normally be REAL but other simple types (e.g., INTEGER, STRING) are possible.

G.3.4.3 data_conversion

data_conversion contains conversion factors for recovering raw dimensional data from given nondimensional data.

Given a nondimensional piece of data, `Data(nondimensional)`, the conversion to 'raw' dimensional form is:

```
Data(raw) = Data(nondimensional)*scale + offset
```

EXPRESS specification:

```
*)
ENTITY data_conversion;
  scale  : REAL;
  offset : REAL;
END_ENTITY;
(*
```

Attribute definitions:

scale: The scaling factor.

offset: The offset.

G.3.4.4 dimensional_units

dimensional_units describes the system of units used to measure dimensional data.

EXPRESS specification:

```
*)
ENTITY dimensional_units;
  mass        : mass_units;
  length      : length_units;
  time        : time_units;
  temperature : temperature_units;
  angle       : angle_units;
END_ENTITY;
(*
```

Attribute definitions:

mass: The unit of mass.

length: The unit of length.

time: The unit of time.

temperature: The unit of temperature.

angle: The unit of plane angle.

G.3.4.5 dimensional_exponents

dimensional_exponents describes the dimensionality of data by specifying the exponents associated with each of the fundamental units of measure.

EXAMPLE 1 An instance of **dimensional_exponents** that describes velocity is:

EXPRESS-I specification:

```
velocity = {mass -> 0.0;
            length -> 1.0;
            time -> -1.0;
            temperature -> 0.0;
            angle -> 0.0;};
```

EXPRESS specification:

```
*)
ENTITY dimensional_exponents;
  mass        : REAL;
  length      : REAL;
  time        : REAL;
  temperature : REAL;
  angle       : REAL;
END_ENTITY;
(*
```

Attribute definitions:

mass: The dimensionality of units of mass.

length: The dimensionality of units of length.

time: The dimensionality of units of time.

temperature: The dimensionality of units of temperature.

angle: The dimensionality of units of angle.

G.3.4.6 index_list

index_list specifies a list of indices.

EXPRESS specification:

```
*)
ENTITY index_list;
  nindices : INTEGER;
  indices   : LIST [1:?] OF ARRAY [1:nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: The number of indices required to reference a node.

indices: the list of indices.

G.3.4.7 index_range

index_range specifies the beginning and ending indices of a subrange.

EXPRESS specification:

```
*)
ENTITY index_range;
 nindices : INTEGER;
 start     : ARRAY [1 : nindices] OF INTEGER;
 finish    : ARRAY [1 : nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: The number of indices required to reference a node.

start: The indices of the minimal corner of the subrange;

finish: The indices of the maximal corner of the subrange.

G.3.4.8 data_array

data_array describes a multi-dimensional data array of a given type, dimensionality and size in each dimension. The data may be dimensional, nondimensional or pure constants. Qualifiers are provided to describe dimensional units or normalization information associated with the data.

The data type will usually be REAL but other simple data types are possible.

This structure is formulated to describe an array of scalars. Therefore, for vector quantities (e.g., a position vector or a velocity vector), seperate instances are required for each component of the vector.

EXAMPLE 1 The cartesian coordinates of a 3-D grid are described by three separate data arrays: one for x, one for y, and one for z.

The optional attributes of **data_array** provide information for manipulating the data, including changing units or normalization. Within a given instance of **data_array**, the class of data and all information required for manipulations may be completely and precisely specified by the values of **class**, **units**, **exponents** and **conversion**. **class** identifies the class of data and governs the manipulations that can be performed.

NOTE 1 Data array is described in detail in ISO 10303-5w.

EXPRESS specification:

```
*)
ENTITY data_array;
  descriptions : OPTIONAL LIST OF textual;
  id           : label;
  dimension    : INTEGER;
  sizes        : ARRAY [1:dimension] OF INTEGER;
  data         : array_of_data;
  classifier   : OPTIONAL data_name;
  units_class  : OPTIONAL data_class;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_data_array FOR data_array;
  ONEOF(dimensional_data_array,
        nondimensional_data_array);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: is annotation;

id: User-specified instance identifier;

dimension: The number of dimensions in the multidimensional data array;

sizes: The array sizes for each dimension;

data: The data values;

classifier: An identifier or name that identifies and describes the quantity being stored;

units_class: The class of data;

G.3.4.9 dimensional_data_array

A **dimensional_data_array** is a **data_array** holding data that is dimensional.

EXPRESS specification:

```
*)
ENTITY dimensional_data_array
  SUBTYPE OF (data_array);
  units        : dimensional_units;
  exponents    : dimensional_exponents;
  conversion   : OPTIONAL data_conversion;
WHERE
  wr1 : NOT EXISTS(SELF/data_array.units_class);
END_ENTITY;
(*
```

Attribute definitions:

units: The dimensional units of the data;

exponents: The dimensional exponents;

conversion: The normalization.

Formal propositions:

wr1: The (inherited) **units_class** attribute shall have no value.

G.3.4.10 nondimensional_data_array

A **nondimensional_data_array** is a **data_array** holding data that is not dimensional.

EXPRESS specification:

```
*)
ENTITY nondimensional_data_array
  SUBTYPE OF (data_array);
  SELF/data_array.units_class : data_class;
WHERE
  wr1 : (units_class <> dimensional) AND
        (units_class <> unspecified);
END_ENTITY;
(*
```


Attribute definitions:

class: The class of data;

Formal propositions:

wr1: The value of the (inherited) **units_class** shall not be either **dimensional** or **unspecified**.

G.3.4.11 rind

rind describes the number of rind planes associated with a data array containing grid coordinates, flow solution data, or any other grid-related discrete data.

EXPRESS specification:

```
*)
ENTITY rind;
  nindices : INTEGER;
  planes   : ARRAY [1:2*nindices] OF INTEGER;
END_ENTITY;
(*
```

Attribute definitions:

nindices: The number of indices required to reference a node.

planes: contains the number of rind planes attached to the minimum and maximum faces of a zone. Further description is given in ISO 10303-5w.

G.3.4.12 geometry_reference

Reference to a geometry data file.

EXPRESS specification:

```
*)
ENTITY geometry_reference;
  descriptions : OPTIONAL LIST OF textual;
  file         : STRING;
END_ENTITY;
(*
```


Attribute definitions:

descriptions: is annotation;

file: is the location of the file.

EXPRESS specification:

```
*)
END_SCHEMA; -- end of basis
(*
```

G.4 domain

The following EXPRESS declaration begins the **domain** schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA domain;
  REFERENCE FROM basis;
  REFERENCE FROM hierarchy;
(*
```

G.4.1 Introduction

This schema defines and describes the structure types for describing the grid coordinates and grid interfaces pertaining to a zone.

The graphical form for the **domain** schema is given in Figures G.17 through G.18.

G.4.2 Fundamental concepts and assumptions

A grid is defined by its vertices. In a 3–D structured grid, the volume is the ensemble of cells, where each cell is the hexahedron region defined by eight nearest neighbor vertices. Each cell is bounded by six faces, where each face is the quadrilateral made up of four vertices. An edge links two nearest-neighbor vertices; a face is bounded by four edges.

In a 2–D structured grid, the notation is more ambiguous. Typically, the quadrilateral area composed of four nearest-neighbor vertices is referred to as a cell. The sides of each cell, the line linking two vertices, is either a face or an edge. In a 1–D grid, the line connecting two vertices is a cell.

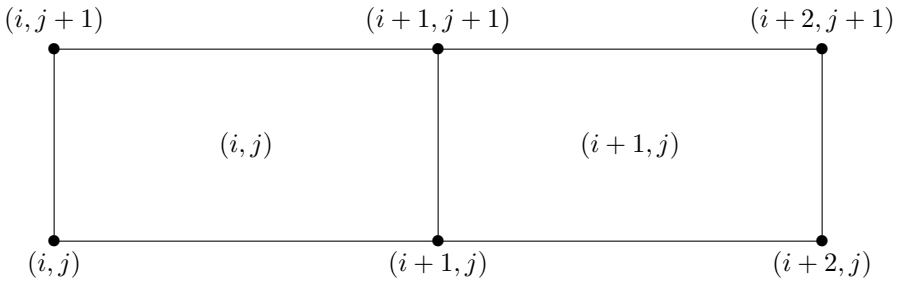


Figure G.12 – Example convention for a 2-D cell center

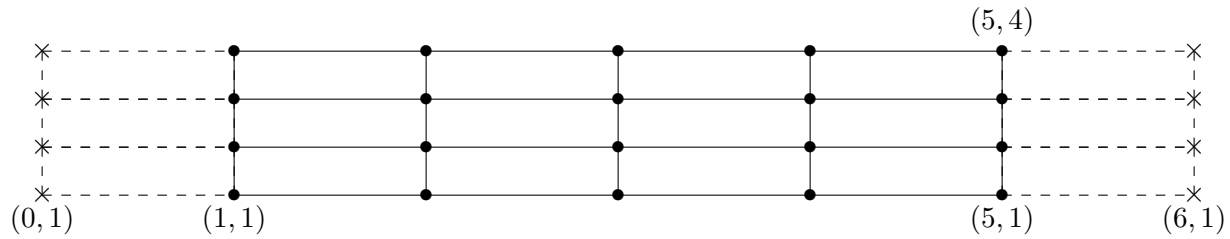


Figure G.13 – Example grid block with rind vertices

A structured-multiblock grid is composed of zones, where each zone includes all the vertices, cells, faces and edges that constitute a grid block.

Indices describing a 3–D grid are ordered (i, j, k); (i, j) is used for 2–D and (i) for 1–D.

Cell centers, face centers, and edge centers are indexed by the minimum of the connecting vertices.

EXAMPLE 1 For example a 2-D cell center (or face center on a 3-D grid) would have the conventions shown in Figure G.12.

In addition, the default beginning vertex for a grid block is $(1, 1, 1)$; this means the default beginning cell center of a grid block is also $(1, 1, 1)$.

A grid block may contain grid-coordinate or flow-solution data defined at a set of points outside the block itself. These are referred to as 'rind' or ghost points and may be associated with fictitious vertices or cell centers. They are distinguished from the vertices and cells making up the grid block (including its boundary vertices), which are referred to as 'core' points.

EXAMPLE 2 Figure G.13 shows a 2–D grid block with a single row of 'rind' vertices at the minimum and maximum i-faces. The grid size (i.e. the number of 'core' vertices in each direction) is 5×4. 'Core' vertices are designated by '•', and 'rind' vertices by '×'. Default indexing is also shown for the vertices.

For a grid (or zone), the minimum faces in each coordinate direction are denoted i-min, j-min and k-min; the maximum faces are denoted i-max, j-max and k-max. These are the minimum and maximum 'core' faces.

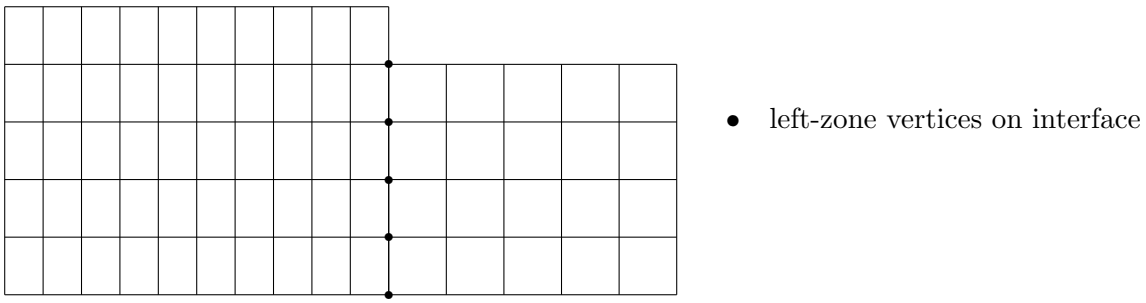


Figure G.14 – A 1-to-1 abutting interface

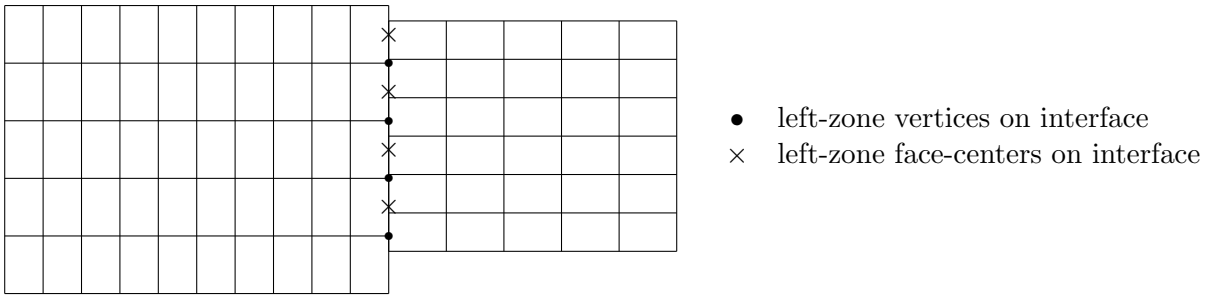


Figure G.15 – A mismatched abutting interface

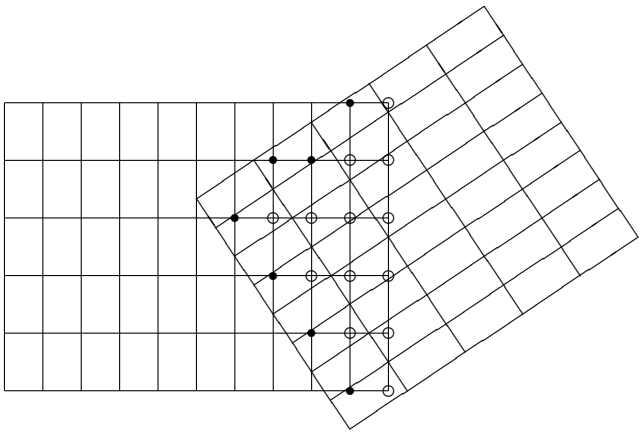
EXAMPLE 3 $i-$min is the face or grid plane whose core vertices have minimum i index (which if using default indexing is 1).

Figures G.14 to Figure G.16 show three types of multiblock interfaces.

Figure G.14 illustrates a 1-to-1 abutting interface, also referred to as matching or C0 continuous. The interface is a plane of vertices that are physically coincident (i.e., they have identical coordinate values) between the adjacent zones; grid-coordinate lines perpendicular to the interface are continuous from one zone to the next. In 3-D, a 1-to-1 abutting interface is always a logically rectangular region.

The second type of interface, is mismatched abutting, where two zones touch but do not overlap (except for vertices and cell faces on the grid plane of the interface). Vertices on the interface may not be physically concident between the two zones. Figure G.15 indentifies the vertices and face centers of the left zone that lie on the interface. In 3-D, the vertices of a zone that constitute an interface patch may not form a logically rectangular.

The third type of multiblock interface is called overset and occurs when two zones overlap; in 3-D, the overlap is a 3-D region. For overset interfaces, one of the two zones takes recedence over the other; this establishes which solution in the overlap region to retain and which one to discard. The region in a given zone where the solution is discarded is called an overset hole and the grid points outlining the hole are called fringe points.



- left-zone fringe points (vertices)
- left-zone overset-hole points (vertices)

Figure G.16 – An overset interface

Figure G.16 depicts an overlap region between two zones, where the right zone takes precedence over the left zone. The points identified in Figure G.16 are the fringe points and overset-hole points for the left zone. In addition, for the zone taking precedence, any bounding points (i.e., vertices on the bounding faces) of the zone that lies within the overlap must also be identified.

Overset interfaces may include multiple layers of fringe points outlining holes and at zone boundaries.

For the mismatched abutting and overset interfaces in Figure G.15 and Figure G.16, the left zone plays the role of receiver zone and the right plays the role of donor zone.

G.4.3 domain entity definitions

G.4.3.1 grid_coordinates

The physical coordinates of the grid vertices in a zone are described by the **grid_coordinates** structure. The structure contains a list for the data arrays of the individual components of the position vector. It also provides a mechanism for identifying rind-point data included within the position-vector arrays.

EXPRESS specification:

```
*)
ENTITY grid_coordinates;
  descriptions      : OPTIONAL LIST OF textual;
  rind              : OPTIONAL rind;
  data              : OPTIONAL LIST OF data_array;
  dclass            : OPTIONAL data_class;
  dimunits          : OPTIONAL dimensional_units;
```

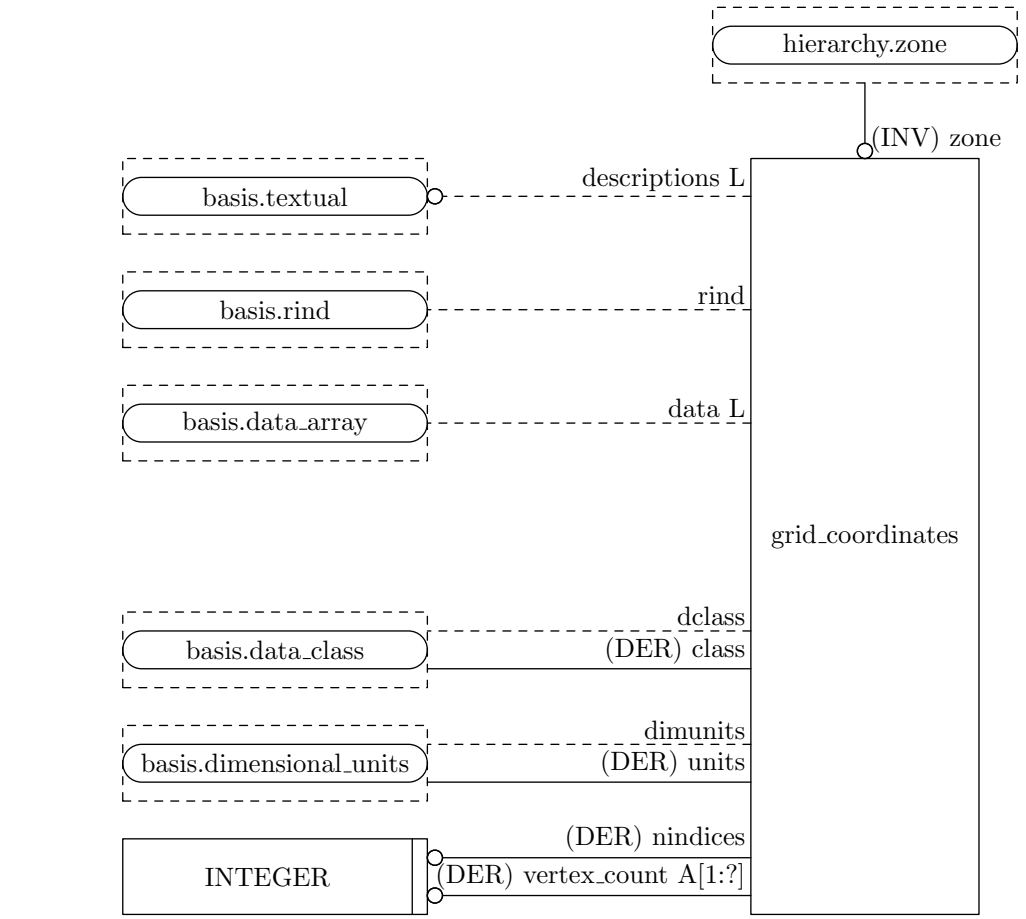



Figure G.17 – Entity level diagram of ARM domain schema (page 1 of 2)

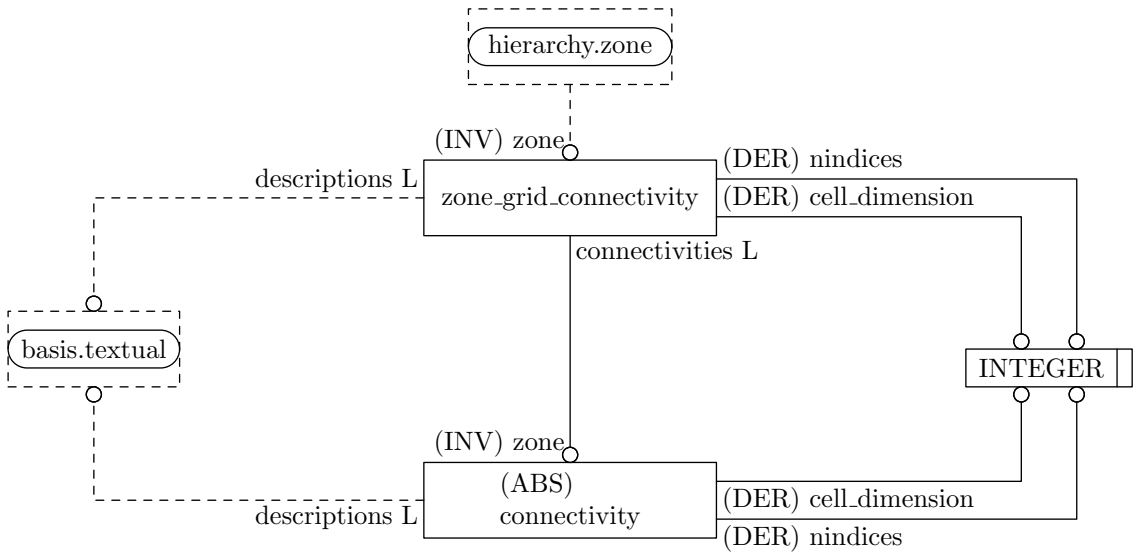


Figure G.18 – Entity level diagram of ARM domain schema (page 2 of 2)


```
DERIVE
  nindices   : INTEGER := zone.nindices;
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  vertex_count : ARRAY [1:nindices] OF INTEGER := zone.vertex_count;
INVERSE
  zone : zone FOR coordinates;
END_ENTITY;
(*
```

Attribute definitions:

descriptions: is annotations;

rind: is optional. If not given then this is equivalent to a **rind** structure whose **planes** array contains all zeros.

data: is the grid-coordinate data; each **data_array** shall contain a single component of the position vector (e.g., three structures are required for 3-D data, one for each coordinate value).

dclass: non-default data class;

dimunits: non-default system of units;

nindices: The number of indices required to reference a node.

class: is the default class for data contained in **data_array**.

units: describes the system of units employed.

vertex_count: is the number of vertices, excluding rind points, in each index direction

zone: is the calling zone.

Informal propositions:

ip1: The **nindices** of **rind** shall match the grid coordinates **nindices**.

ip2: Grid coordinates for an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

ip3: The **data** shall be consistent.

G.4.3.2 zone_grid_connectivity

All multiblock interface grid connectivity interface information pertaining to a given zone is contained in the **zone_grid_connectivity** structure. This includes abutting interfaces (general mismatched and 1-to-1), overset-grid interfaces, and overset-grid holes.

EXPRESS specification:

```
*)
ENTITY zone_grid_connectivity;
  descriptions    : OPTIONAL LIST OF textual;
  connectivities : LIST OF connectivity;
DERIVE
  nindices     : INTEGER := zone.nindices;
  cell_dimension  : INTEGER := zone.cell_dimension;
INVERSE
  zone : zone FOR grid_connectivity;
END_ENTITY;
(*
```

Attribute definitions:

descriptions: is annotation;

connectivities: is the connectivity information;

nindices: The number of indices required to reference a node.

cell_dimension: is the dimension of a cell in the mesh.

zone: is the zone.

G.4.3.3 connectivity

Information specifying the connectivity of a multiblock interface.

All the interface patches for a given zone are contained in the **zone_grid_connectivity** entity for that zone. If a face of a zone touches several other zones (say N), the N different instances of the **connectivity** structure must be included in the zone to describe each interface patch.

NOTE 1 This convention requires that a single interface patch be described twice in the database — once for each adjacent zone. It also means that the database is symmetrical with regard to interface patches.

NOTE 2 Connectivity is described in detail in ISO 10303-5w.

EXPRESS specification:

```
*)
ENTITY connectivity;
  descriptions : OPTIONAL LIST OF textual;
DERIVE
  nindices       : INTEGER := zone.nindices;
  cell_dimension  : INTEGER := zone.cell_dimension;
```



```
INVERSE
  zone : zone_grid_connectivity FOR connectivities;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_connectivity FOR connectivity;
  ABSTRACT SUPERTYPE;
--  ONEOF(grid_connectivity_1to1,
--        grid_connectivity,
--        overset_holes);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: is annotation;

nindices: The number of indices required to reference a node.

cell_dimension: Dimension of a cell in the mesh.

zone: is the calling **zone_grid_connectivity**.

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of domain
(*
```

G.5 conditions

The following EXPRESS declaration begins the **conditions** schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA conditions;
  REFERENCE FROM basis;
  REFERENCE FROM hierarchy;
  REFERENCE FROM domain (elements);
(*
```

G.5.1 Introduction

This schema defines and describes the boundary-condition specifications within Navier-Stokes codes.

The graphical form for the **conditions** schema is given in Figures G.20 through G.27.

G.5.2 Fundamental concepts and assumptions

This model is an attempt to unify boundary-condition specifications within Navier-Stokes codes. The structures and conventions presented are a compromise between simplicity and generality.

The difficulty with boundary-conditions is that there is such a wide variety used, and even a single boundary-condition equation is often implemented differently in different codes. Some boundary-conditions, such as a symmetry plane, are fairly well defined. Other boundary-conditions are much looser in their definition and implementation. An inflow boundary is such an example. It is generally accepted how many solution quantities should be specified at an inflow boundary (from mathematical well-posedness arguments), but what those quantities are will change with the class of flow problems (e.g., internal flows *vs.* external flows), and they will also change from code to code.

An additional difficulty for CFD analysis is that in some situations different boundary-condition equations are applied depending on local flow conditions. Any boundary where the flow can change from inflow to outflow or supersonic to subsonic is a candidate for flow-dependent boundary-condition equations.

These difficulties have moulded the design of the boundary-condition structures and conventions. Boundary-condition types are defined (**bc_type_simple**, **bc_type_compound**, and **bc_type**) that establish the equations to be enforced. However, for those more loosely defined boundary-conditions, such as inflow/outlow, the boundary-condition type merely establishes general guidelines on the equations to be imposed. Augmenting (and superseding) the information provided by the boundary-condition type is precisely defined boundary-condition solution data. Data-name conventions (**data_name**) are used to identify the exact quantities involved in the boundary-conditions.

One flexibility that is provided by this approach is that boundary-condition information can easily be built during the course of analysis. For example, during grid-generation phases minimal information (e.g., the boundary-condition type) may be given. Then, prior to running of the flow solver, more specific boundary-condition information, such as Dirichlet or Neumann data, may be added.

An additional flexibility proved by the structures is that both uniform and non-uniform boundary-condition data can be described within the same framework.

Boundary-conditions are classified as either fixed or flow-dependent. Fixed boundary-conditions enforce a given set of boundary-condition equations regardless of flow conditions; flow-dependent boundary-conditions enforce different sets of boundary-condition equations depending on local flow conditions. Both fixed and flow-dependent boundary-conditions are incorporated into a uniform framework, which allows all boundary-conditions to be described in a similar manner.

Figure G.19 depicts the hierarchy used for prescribing a single boundary-condition. Each

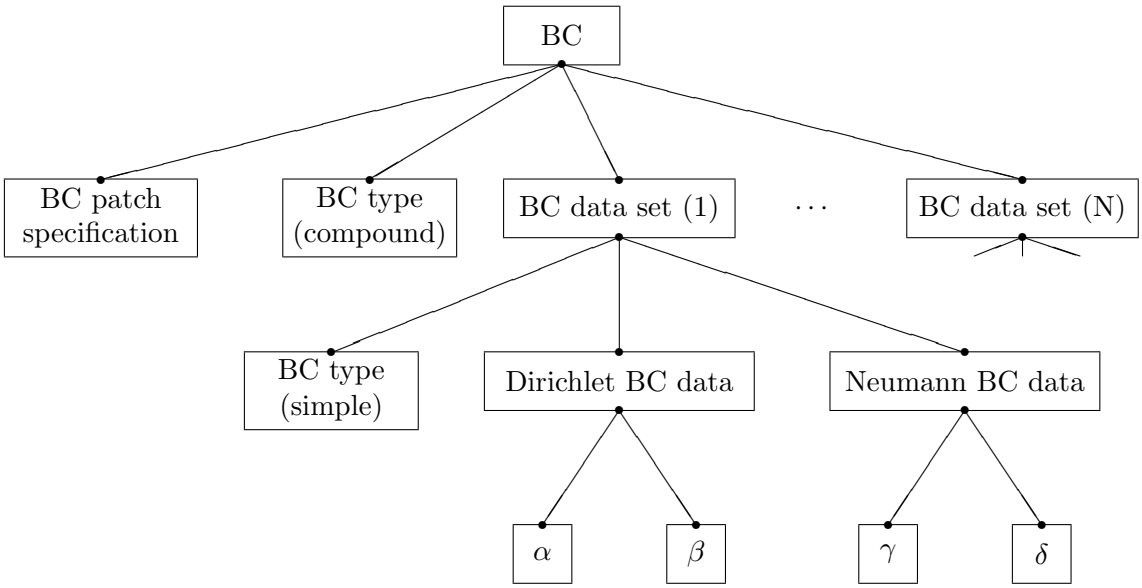


Figure G.19 – Hierarchy for boundary-condition structures

boundary-condition includes a type that describes the general equations to enforce, a patch specification, and a collection of data sets. The minimum required information for any boundary-condition is the patch specification and the boundary-condition type. This minimum information is similar to that used in many existing flow solvers.

Generality in prescribing equations to enforce and their associated boundary-condition data is provided in the optional data sets. Each data set contains all boundary-condition data required for a given fixed or simple boundary-condition. Each data set is also tagged with a boundary-condition type. For fixed boundary-conditions, the hierarchical tree contains a single data set, and the two boundary-condition types shown in Figure G.19 are identical. Flow-dependent or compound boundary-conditions contain multiple data sets, each to be applied seperately depending on local flow conditions. The compound boundary-condition type describes the general flow-dependent boundary-conditions, and each data set contains associated simple boundary-condition types. For example, a farfiled boundary condition would contain for data sets, where each applies to the different combinations of subsonic and supersonic inflow and outflow.

Within a single data set, boundary-condition data is grouped by equation type into Dirichlet and Neumann data. The lower leaves of Figure G.19 show data for generic flow-solution quantities α and β to be applied in Dirichlet conditions, and data for γ and δ to be applied in Neumann boundary-conditions. **data_array** entities are employed to store these data and to identify the specific flow variables they are associated with.

In situations where the data sets (or any information contained therein) are absent from a given boundary-condition hierarchy, flow solvers are free to impose any appropriate boundary-conditions. Although not pictured in Figure G.19, it is also possible to specify the reference state from which the flow solver should extract the boundary-condition data.

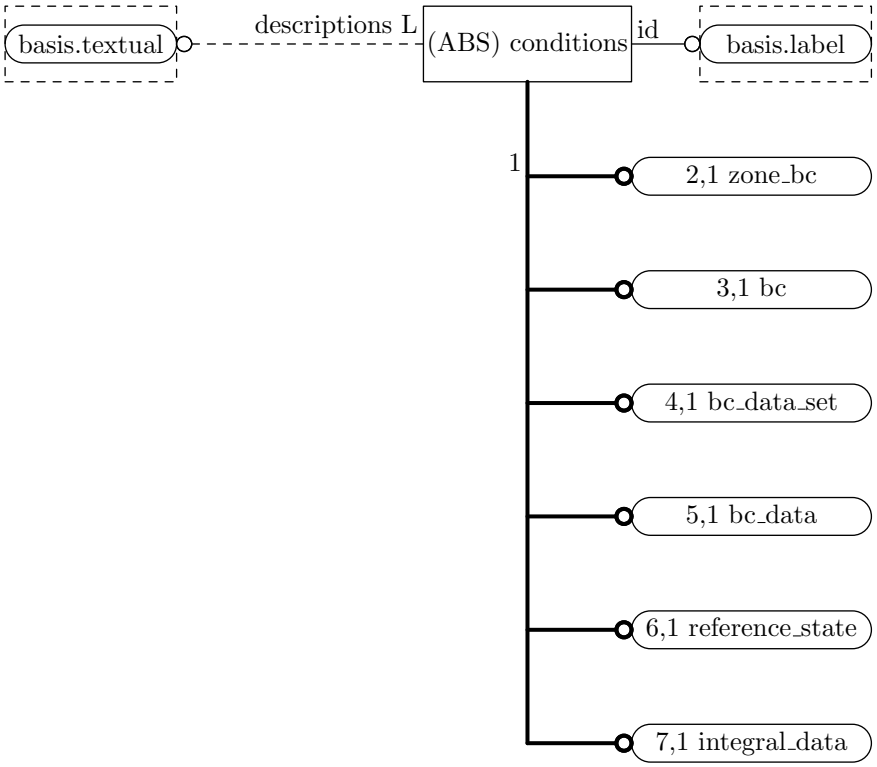


Figure G.20 – Entity level diagram of ARM conditions schema (page 1 of 8)

G.5.3 conditions type definitions

G.5.3.1 Riemann_1D_data_name

Riemann_1D_data_name is an enumeration of standardized Riemann data for 1-D flow.

Boundary condition specification for inflow/outflow or farfield boundaries often involves Riemann invariants or characteristics of the linearized inviscid flow equations. For an ideal compressible gas, these are typically defined as follows: Riemann invariants for an isentropic 1–D flow are,

$$\left[\frac{\partial}{\partial t} + (u \pm c)\frac{\partial}{\partial x}\right]\left(u \pm \frac{2}{\gamma - 1}c\right) = 0.$$

Characteristic variables for the 3–D Euler equations linearized about a constant mean flow are,

$$\left[\frac{\partial}{\partial t} + \bar{\Lambda}_n \frac{\partial}{\partial x}\right] W_n'(x,t) = 0, \qquad n = 1, 2, \ldots 5,$$

where the characteristics and corresponding characteristic variables are

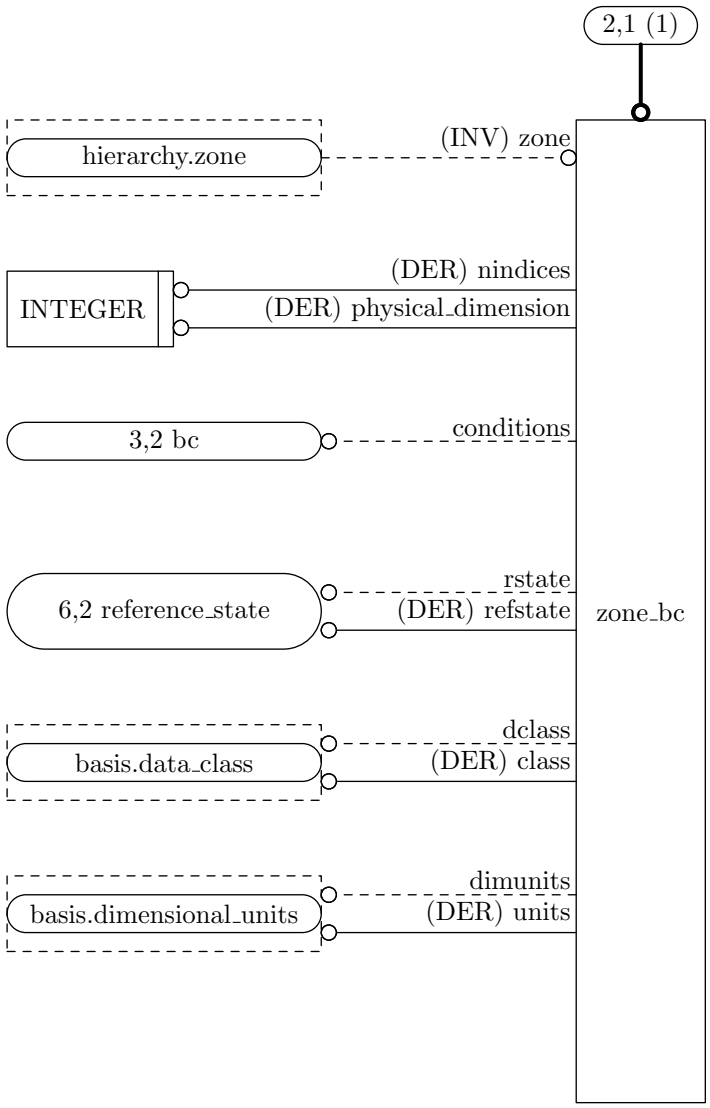


Figure G.21 – Entity level diagram of ARM conditions schema (page 2 of 8)

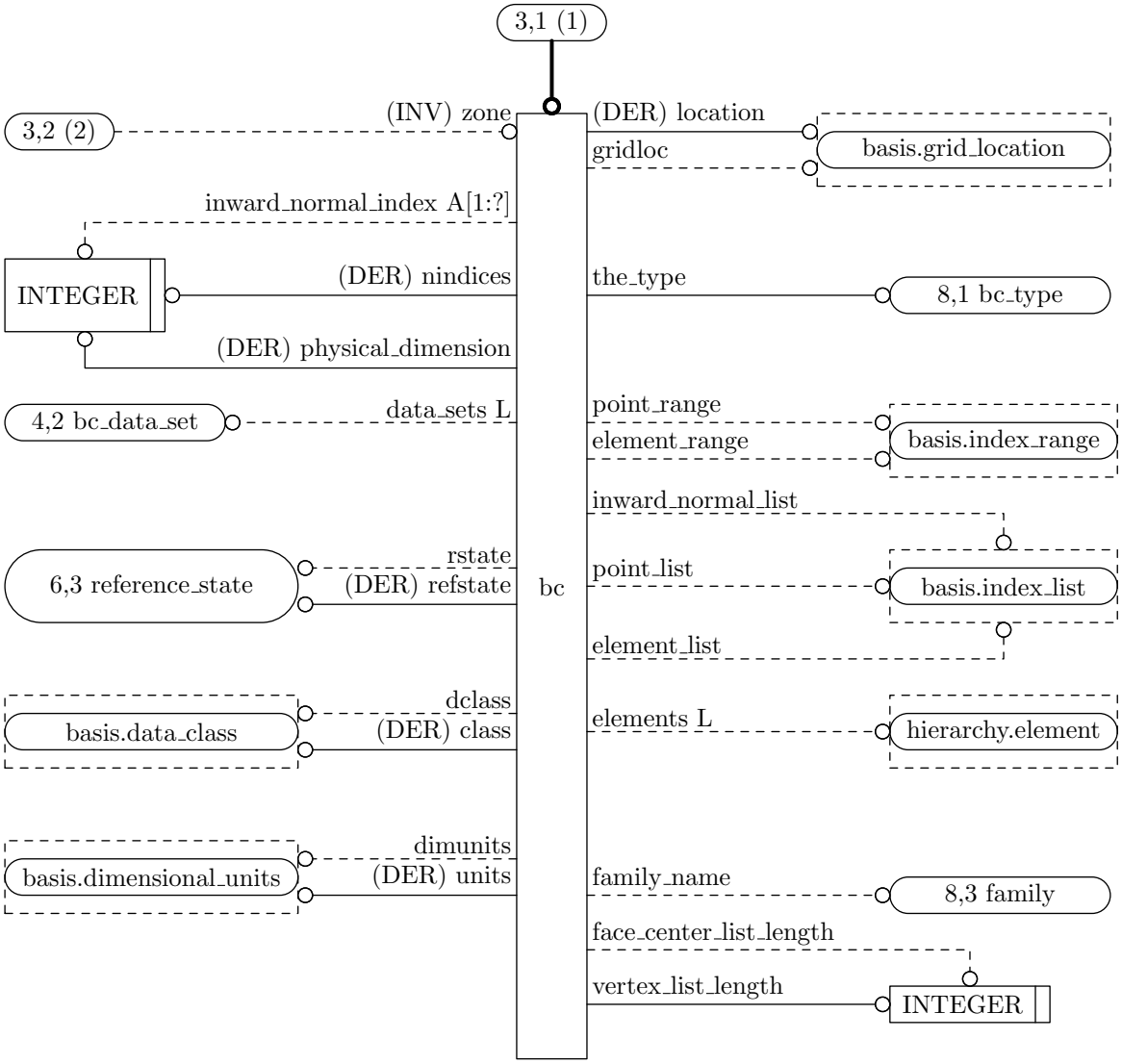


Figure G.22 – Entity level diagram of ARM conditions schema (page 3 of 8)

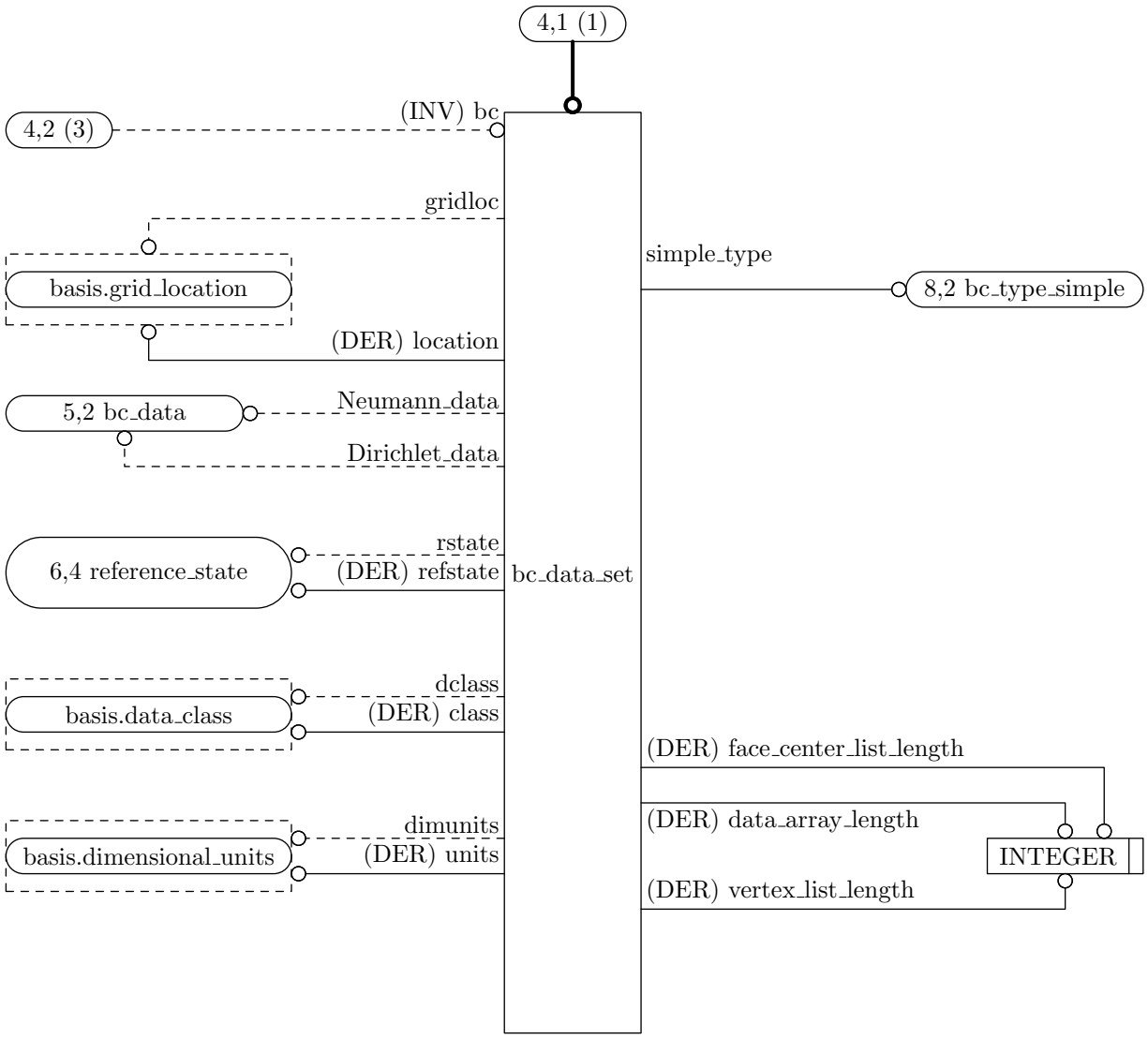


Figure G.23 – Entity level diagram of ARM conditions schema (page 4 of 8)

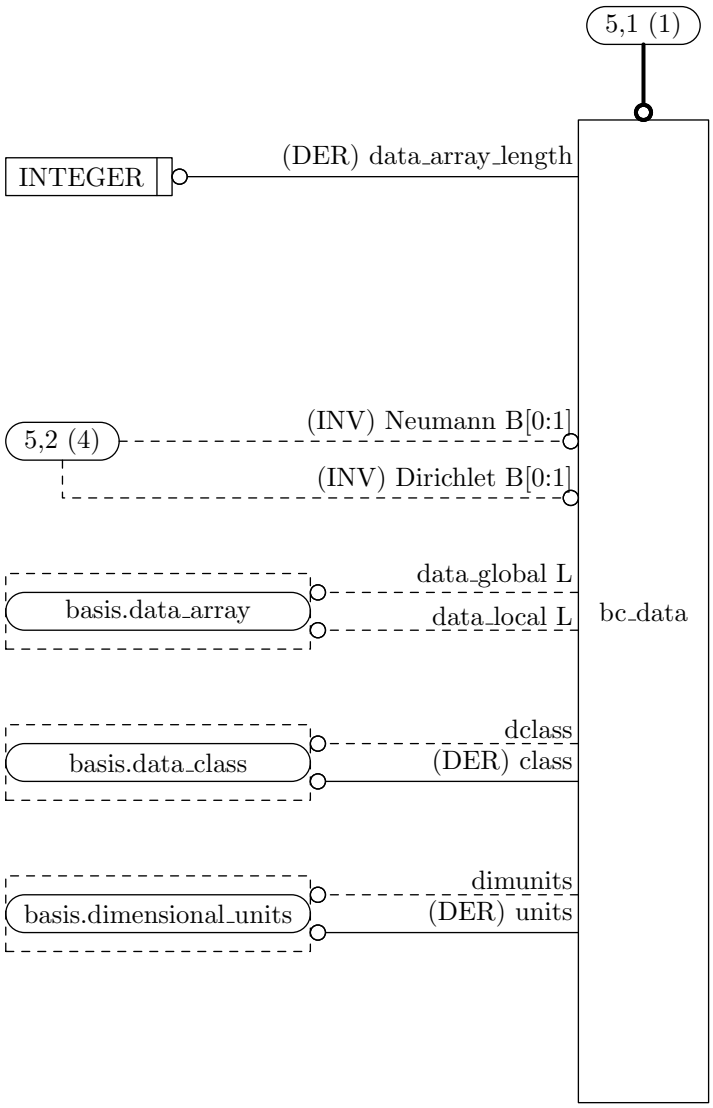


Figure G.24 – Entity level diagram of ARM conditions schema (page 5 of 8)

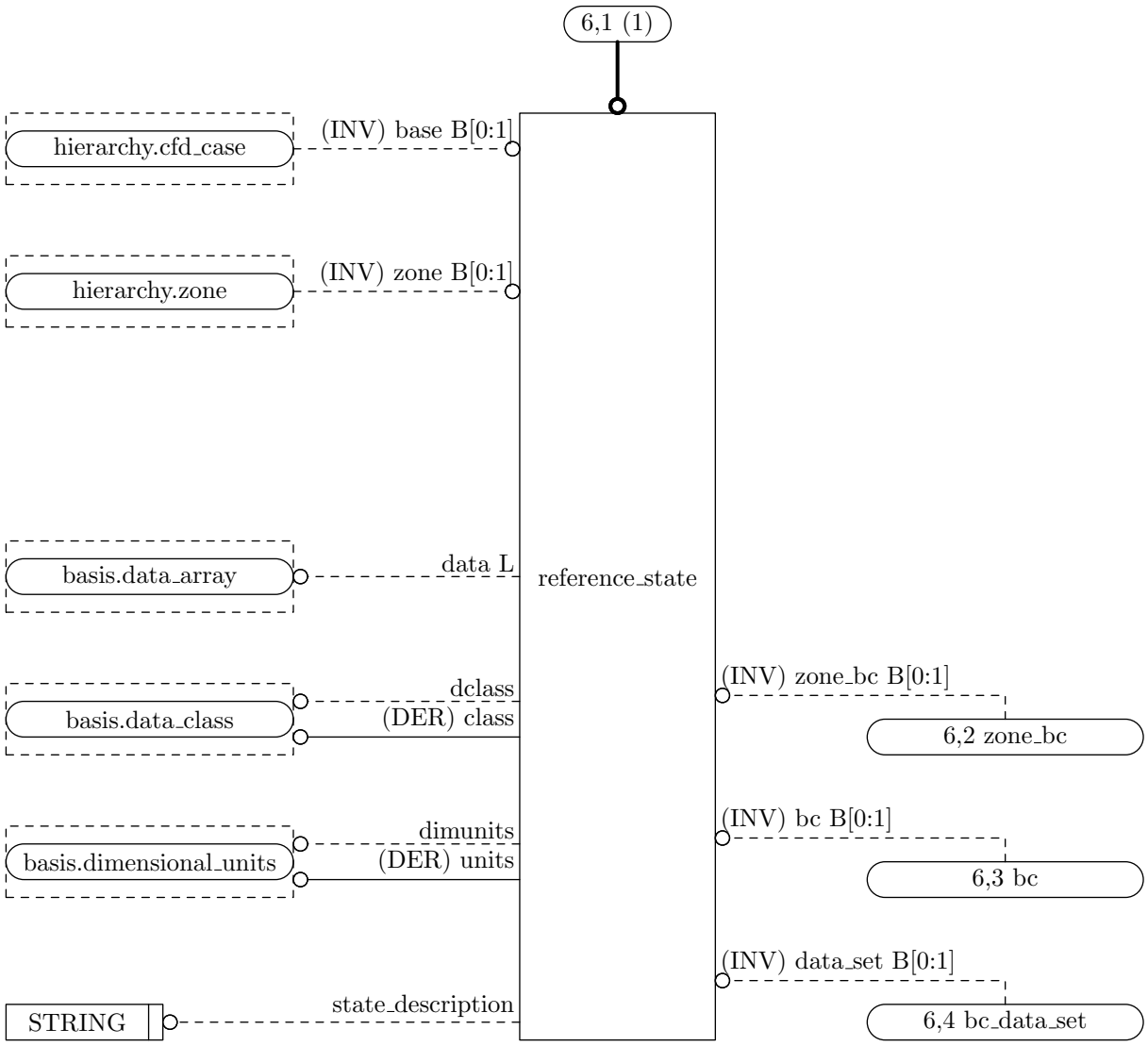


Figure G.25 – Entity level diagram of ARM conditions schema (page 6 of 8)

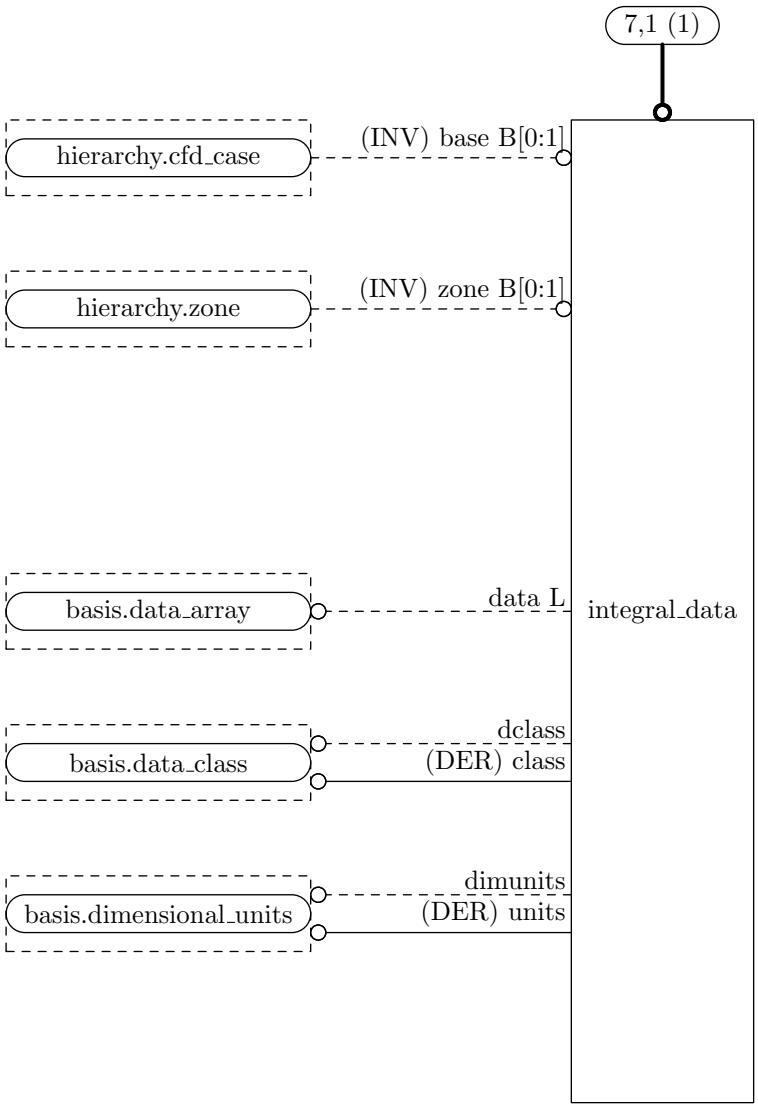


Figure G.26 – Entity level diagram of ARM conditions schema (page 7 of 8)

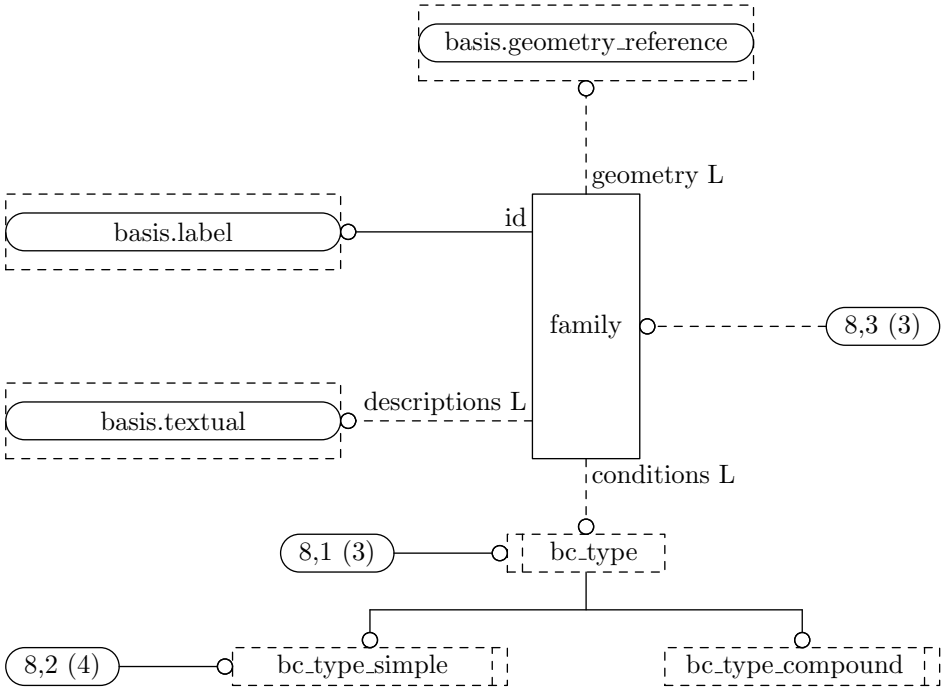


Figure G.27 – Entity level diagram of ARM conditions schema (page 8 of 8)

Characteristic	$\bar{\Lambda}_n$	W'_n
'entropy'	$\bar{u}$	$p' - \rho'/\bar{c}^2$
'vorticity'	$\bar{u}$	v'
'vorticity'	$\bar{u}$	w'
'acoustic'	$\bar{u} \pm \bar{c}$	$p' \pm u'/(\bar{\rho}\bar{c})$

Barred quantities are evaluated at the mean flow, and primed quantities are linearized perturbations. The only non-zero mean-flow velocity component is $\bar{u}$.

EXPRESS specification:

```
*)
TYPE Riemann_1D_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

The required identifiers and their meanings are given in Table 9.

G.5.3.2 bc_type

Boundary-condition types identify the equations that should be enforced at a given boundary location. The boundary-condition types are described by **bc_type**. Some members of **bc_type**

completely identify the equations to impose, while others identify a general description of the class of boundary-condition equations to impose.

bc_type is subdivided into two enumeration types: **bc_type_simple** and **bc_type_compound** which identify the simple and compound boundary-condition types respectively.

The subdivision of **bc_type** is based on function. For simple boundary-conditions, the equations and data are fixed; whereas, for compound boundary-conditions different sets of equations are imposed depending on local flow conditions at the boundary.

bc_type is a superset of **bc_type_simple** and **bc_type_compound**. It identifies the boundary-condition (simple or compound) at a boundary location.

For inflow/outflow boundary-condition descriptions, 3-D inviscid compressible flow is assumed; the 2-D equivalent should be obvious. These same boundary-conditions are typically used for viscous cases also. This '3-D Euler' assumption will be noted wherever used.

EXPRESS specification:

```
*)
TYPE bc_type = SELECT
     (bc_type_simple,
      bc_type_compound);
END_TYPE;
(*
```

G.5.3.3 bc_type_simple

bc_type_simple is an enumeration type that identifies the simple boundary-condition at a boundary location.

In the descriptions below, Q is the solution vector, $\vec{q}$ is the velocity vector whose magnitude is q, the unit normal to the boundary is $\hat{n}$, and $\partial()/\partial n = \hat{n} \cdot \nabla$ is differentiation normal to the boundary.

EXPRESS specification:

```
*)
TYPE bc_type_simple = ENUMERATION OF
     (unspecified,
      user_defined,
      bc_general,
      bc_Dirichlet,
      bc_Neumann,
      bc_extrapolate,
```



```
      bc_wall_inviscid,
      bc_wall_viscous_heat_flux,
      bc_wall_viscous_isothermal,
      bc_wall_viscous,
      bc_wall,
      bc_inflow_subsonic,
      bc_inflow_supersonic,
      bc_outflow_subsonic,
      bc_outflow_supersonic,
      bc_tunnel_inflow,
      bc_tunnel_outflow,
      bc_degenerate_line,
      bc_degenerate_point,
      bc_symmetry_plane,
      bc_symmetry_polar,
      bc_axissymmetric_wedge);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: condition is unspecified;

user_defined: condition is specified via an external agreement between the data creator and the data user;

bc_general: arbitrary conditions on Q or $\partial Q/\partial n$;

bc_Dirichlet: Dirichlet condition on Q vector;

bc_Neumann: Neumann condition on $\partial Q/\partial n$;

bc_extrapolate: extrapolate Q from interior;

bc_wall_inviscid: inviscid (slip) wall

— normal velocity specified (default: $\vec{q} \cdot \hat{n} = 0$)

bc_wall_viscous_heat_flux: viscous no-slip wall with heat flux

— velocity Dirichlet (default: $q = 0$)

— temperature Neumann (default: adiabatic, $\partial T/\partial n = 0$)

bc_wall_viscous_isothermal: viscous no-slip, isothermal wall

— velocity Dirichlet (default: $q = 0$)

— temperature Dirichlet

bc_wall_viscous: viscous no-slip wall; special cases are **bc_wall_viscous_heat_flux** and **bc_wall_viscous_isothermal**.

— velocity Dirichlet (default: $q = 0$)

— Dirichlet or Neumann on temperature

bc_wall: general wall condition; special cases are **bc_wall_inviscid**, **bc_wall_viscous**, **bc_wall_viscous_heat_flux**, and **bc_wall_viscous_isothermal**

bc_inflow_subsonic: inflow with subsonic normal velocity

— specify 4; extrapolate 1 (3-D Euler)

bc_inflow_supersonic: inflow with supersonic normal velocity

— specify 5; extrapolate 0 (3-D Euler)

same as **bc_Dirichlet**

bc_outflow_subsonic: outflow with subsonic normal velocity

— specify 1; extrapolate 0 (3-D Euler)

bc_outflow_supersonic: outflow with supersonic normal velocity

— specify 0; extrapolate 5 (3-D Euler)

same as **bc_Extrapolate**

bc_tunnel_inflow: tunnel inlet (subsonic normal velocity)

— specify cross-flow velocity, stagnation enthalpy, entropy

— extrapolate 1 (3-D Euler)

bc_tunnel_outflow: tunnel exit (subsonic normal velocity)

— specify static pressure

— extrapolate 4 (3-D Euler)

bc_degenerate_line: face degenerated to a line;

bc_degenerate_point: face degenerated to a point;

bc_symmetry_Plane: symmetry plane; face should be coplanar

— density, pressure: $\partial()/\partial n = 0$

— tangential velocity: $\partial(\vec{q} \times \hat{n})/\partial n = 0$

— normal velocity: $\vec{q} \cdot \hat{n} = 0$

bc_symmetry_polar: polar-coordinate singularity line; special case of **bc_degenerate_line** where degenerate face is a straight line and flowfield has polar symmetry; $\hat{s}$ is singularity line tangential unit vector

— normal velocity: $\vec{q} \times \hat{s} = 0$

— all others: $\partial()/\partial n = 0$, n normal to $\hat{s}$

bc_axissymmetric_wedge: axisymmetric wedge; special case of **bc_degenerate_line** where degenerate face is a straight line

G.5.3.4 bc_type_compound

bc_type_compound is an enumeration type that identifies the compound boundary-condition at a boundary location.

EXPRESS specification:

```
*)
TYPE bc_type_compound = ENUMERATION OF
     (unspecified,
      user_defined,
      bc_inflow,
      bc_outflow,
      bc_farfield);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: condition is unspecified;

user_defined: condition is specified via an external agreement between the data creator and the data user;

bc_inflow: inflow, arbitrary normal Mach
test on normal Mach, then perform one of: **bc_inflow_subsonic**, **bc_inflow_supersonic**;

bc_outflow: outflow, arbitrary normal Mach
test on normal Mach, then perform one of: **bc_outflow_subsonic**, **bc_outflow_supersonic**;

bc_farfield: farfield inflow/outflow, arbitrary normal Mach
test on normal velocity and normal Mach, then perform one of: **bc_inflow_subsonic**, **bc_inflow_supersonic**, **bc_outflow_subsonic**, **bc_outflow_supersonic**.

G.5.4 conditions entity definitions

G.5.4.1 condition

A **condition** represents the concept of conditions or constraints pertinent to to an analysis.

EXPRESS specification:

```
*)
ENTITY condition;
  descriptions : OPTIONAL LIST OF textual;
  id           : label;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_condition FOR condition;
  ABSTRACT SUPERTYPE;
  ONEOF(zone_bc,
        bc,
        bc_data_set,
        bc_data,
        reference_state,
        integral_data);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: is annotations;

id: User-specified instance identifier;

G.5.4.2 zone_bc

All boundary-condition information pertaining to a given zone is contained in the **zone_bc** structure.

EXPRESS specification:

```
*)
ENTITY zone_bc
  SUBTYPE OF (condition);
  conditions  : OPTIONAL LIST OF bc;
  rstate      : OPTIONAL reference_state;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  nindices          : INTEGER := zone.nindices;
  physical_dimension : INTEGER := zone.physical_dimension;
  class              : data_class := NVL(dclass, zone.class);
  units              : dimensional_units := NVL(dimunits, zone.units);
  refstate           : reference_state := NVL(rstate, zone.refstate);
INVERSE
  zone : zone FOR conditions;
```



```
END_ENTITY;
(*
```

Attribute definitions:

conditions: is the boundary-conditions for a zone, on a patch by patch basis. Boundary-condition information for a single patch is contained in the **bc** structure. If a zone contains N boundary-condition patches, then N seperate instances of **bc** shall be provided in the **zone_bc** entity for the zone.

rstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

nindices: The number of indices required to reference a node.

physical_dimension: The number of coordinates required to define a node position.

refstate: is reference data applicable to all the boundary-condition of the zone. Reference state data is useful for situations where boundary-condition data is not provided, and flow solvers are free to enforce any appropriate boundary-condition equations.

EXAMPLE 1 An engine nozzle exit boundary-condition usually imposes a stagnation pressure (or some other stagnation quantity) different from freestream. The nozzle-exit stagnation quantities could be specified by **refstate** at this level or below in lieu of providing explicit Dirichlet or Neumman data.

class: is the zonal default for the class of data contained in the zone's boundary-conditions.

units: is the system of dimensional units.

zone: is the zone.

G.5.4.3 bc

bc contains boundary-condition information for a single BC surface patch of a zone. A BC patch is the subrange of the face of a zone where a given boundary-condition is applied.

The structure contains a boundary-condition type, as well as one or more sets of boundary-condition data that are used to define the boundary-condition equations to be enforced on the BC patch. For most boundary-conditions, a single data set is all that is needed. The structure also contains information describing the normal vector to the BC surface patch.

EXPRESS specification:

```
*)
ENTITY bc
  SUBTYPE OF (condition);
  the_type                : bc_type;
```



```
  gridloc                 : OPTIONAL grid_location;
  point_range             : OPTIONAL index_range;
  point_list              : OPTIONAL index_list;
  elements                : OPTIONAL LIST OF elements;
  element_range           : OPTIONAL index_range;
  element_list            : OPTIONAL index_list;
  inward_normal_index     : OPTIONAL ARRAY [1:nindices] OF INTEGER;
  inward_normal_list      : OPTIONAL index_list;
  data_sets               : OPTIONAL LIST OF bc_data_set;
  family_name             : OPTIONAL family;
  rstate                  : OPTIONAL reference_state;
  dclass                  : OPTIONAL data_class;
  dimunits                : OPTIONAL dimensional_units;
  vertex_list_length      : INTEGER;
  face_center_list_length : OPTIONAL INTEGER;
DERIVE
  nindices           : INTEGER := zone.nindices;
  physical_dimension : INTEGER := zone.physical_dimension;
  location           : grid_location := NVL(gridloc, vertex);
  refstate           : reference_state := NVL(refstate, zone.refstate);
  class              : data_class := NVL(dclass, zone.class);
  units              : dimensional_units := NVL(dimunits, zone.units);
INVERSE
  zone : zone_bc FOR conditions;
END_ENTITY;
(*
```

Attribute definitions:

the_type: is the type of the boundary-condition;

gridloc: is non-default location information;

location: is the location of the conditions, which are either at the cell vertices or on the cell faces.

point_range: is a face subrange (i.e., points in a single computational plane); by convention the indices refer to vertices;

point_list: is a face subrange (i.e., points in a single computational plane); by convention the indices refer to vertices;

elements: is the elements;

element_range: is an element subrange;

element_list: is the element indices;

inward_normal_index: shall have only one non-zero element, whose sign indicates the computational-coordinate direction of the BC patch normal; this normal points into the interior of the zone.

Some boundary-conditions require a normal direction to be specified in order to be properly imposed. A computational-coordinate normal can be derived from **point_range** or **point_-**

Table G.1 – inward_normal_index values

Face	Value	Face	Value
i-min	$[+1, 0, 0]$	i-max	$[-1, 0, 0]$
j-min	$[0, +1, 0]$	j-max	$[0, -1, 0]$
k-min	$[0, 0, +1]$	k-max	$[0, 0, -1]$

list by examining redundant index components. Alternatively, this information can be provided directly by **inward_normal_index**. For exterior faces of a zone in 3-D, **inward_normal_index** takes one of the values given in Table G.1.

inward_normal_list: is a list of vectors normal to the BC patch pointing into the interior of the zone; the vectors are not required to be unit vectors. By convention the vectors are located at the vertices of the BC patch.

The physical-space normal vectors of the BC patch may be described by **inward_normal_list**; these are located at vertices, consistent with **point_range** and **point_list**. **inward_normal_list** is specified as an optional attribute because it is not always needed to enforce boundary-conditions, and the physical-space normals of a BC patch can usually be constructed from the grid. However, there are some situations, such as grid-coordinate singularity lines, where **inward_normal_list** becomes a required attribute because the normals cannot be generated from other information.

data_sets: is a list of boundary-condition data sets. In general, the proper **bc_data_set** instance(s) to impose on the BC patch is determined by the value of **the_type**.

For a few boundary-conditions, such as a symmetry plane or polar singularity, the value of **the_type** completely describes the equations to impose, and no instances of **bc_data_set** are needed. For 'simple' boundary-conditions, where a single set of Dirichlet and/or Neumann data is applied a single **bc_data_set** will likely be used (althought this is not a requirement). For 'compound' boundary-conditions, where the equations to impose are dependent on local flow conditions, several instances of **bc_data_set** will likely be used.

refstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

refstate: is reference data applicable to the conditions of the BC patch.

class: is the class of data;

units: is the system of units;

nindices: The number of indices required to reference a node.

physical_dimension: The number of coordinates required to define a node position.

vertex_list_length: is the number of vertices making up the BC patch. If **point_range** is specified, then the value may be determined from the number of grid points (inclusive) between

the beginning and ending indices of **point_range**. If **point_list** is specified then the value is a user input. **vertex_list_length** is also the number of elements in the list **inward_normal_list**.

face_center_list_length: is the number of cell faces making up the BC patch. If **point_range** has a value, then the value of **face_center_list_length** can be easily determined. If the BC patch is not logically rectangular (i.e., if **point_list** is specified), then the value of **face_center_list_length** cannot be determined and has to be a user input.

zone: is the calling **zone_bc**.

Informal propositions:

ip1: One and only one of the following attributes shall have a value: **point_range**, **point_list**, **elements**, **element_range**, **element_list**.

ip2: If **point_range** has a value it shall have the given value for **dimension**.

ip3: If **point_list** has a value, then it shall have the given value of **dimension**.

ip4: If **inward_normal_list** has a value, then it shall have the given value of **dimension**.

ip5: **inward_normal_index** shall have a single nonzero entry;

ip6: If **point_range** and **inward_normal_list** are specified, then a an ordering convention is needed for indices on the BC patch. An ordering convention is also needed if **point_range** is specified and local data is present in the **bc_data_set** substructures. FORTRAN multidimensional array ordering shall be used.

G.5.4.4 family

EXPRESS specification:

```
*)
ENTITY family;
  descriptions : OPTIONAL LIST OF textual;
  id           : label;
  conditions   : OPTIONAL LIST OF bc_type;
  geometry     : OPTIONAL LIST OF geometry_reference;
END_ENTITY;
(*
```

Attribute definitions:

descriptions: is annotation;

id: User-specified instance identifier;

conditions: the family's boundary conditions;

geometry: the family's geometric information;

G.5.4.5 bc_data_set

bc_data_set contains Dirichlet and Neumann data for a single set of boundary-condition equations. Its intended use is for simple boundary-condition types, where the equations imposed do not depend on local flow conditions.

Boundary-condition data is seperated by equation type into Dirichlet and Neumann conditions. Dirichlet boundary-conditions impose the value of the given variables, whereas Neumann boundary-conditions impose the normal derivative of the given variables.

The **bc** structure (clause G.5.4.3) allows for an arbitrary list of boundary-condition data sets, described by the **bc_data_set** structure. For simple boundary-conditions, a single data set must be chosen from a list that may contain more than one element. Likewise, for a compound boundary-condition, a limited number of data sets must be chosen and applied appropriately. The mechanism for proper choice of data sets is controlled by the **the_type** attribute of the **bc** structure, the **the_type** attribute of the **bc_data_set** structure, and the boundary-condition type association table (Table G.2).

bc is used for both simple and compound boundary-conditions; hence, the attribute **bc.the_type** is of type **bc_type**. Conversely, the structure **bc_data_set** is intended to enforce a single set of boundary-condition equations independent of local flow conditions (i.e., it is appropriate only for simple boundary-conditions). That is why the attribute **bc_data_set.simple_type** is of type **bc_type_simple** and not **bc_type**. The appropriate choice of data sets is determined by matching the value of **bc.the_type** with the value of **bc_data_set.simple_type** as specified in Table G.2.

Although the model has a strict division between the two categories of boundary-condition types, in practice some overlap may exist. For example, some of the more general boundary-condition types, such as **bc_wall**, may include a situation of inflow/outflow (for instance if the wall is porous). These complications require further guidelines on appropriate definition and use of boundary-condition types. The real distinctions between **bc_type_simple** and **bc_type_compound** are as follows:

— **bc_type_simple** identifiers always match themselves; **bc_type_compound** never match themselves.

— **bc_type_simple** identifiers always produce a single match; **bc_type_compound** produce multiple matches.

— The usage rule for **bc_type_simple** identifiers is always trivial — apply the single matching data set regardless of local flow conditions.

Therefore, any boundary-condition that involves application of different data sets depending on local flow conditions should be classified as **bc_type_compound**.

NOTE 1 If a type that is classified **bc_type_simple** is desired to be used as a compound (**bc_wall**

Table G.2 – Associated boundary-condition types and usage rules

bc_type Identifier	**Associated bc_type_simple identifiers and usage rules**
bc_inflow	**bc_inflow_supersonic** **bc_inflow_subsonic** *usage rule:* if supersonic normal Mach, choose **bc_inflow_supersonic**, else choose **bc_inflow_subsonic**.
bc_Outflow	**bc_outflow_supersonic** **bc_outflow_subsonic** *usage rule:* if supersonic normal Mach, choose **bc_outflow_supersonic**, else choose **bc_outflow_subsonic**.
bc_farfield	**bc_inflow_supersonic** **bc_inflow_subsonic** **bc_outflow_supersonic** **bc_outflow_subsonic** *usage rule:* if inflow and supersonic normal Mach, choose **bc_inflow_supersonic**, else if inflow, choose **bc_inflow_subsonic**, else if outflow and supersonic normal Mach, choose **bc_outflow_supersonic**, else, choose **bc_outflow_subsonic**.
bc_inflow_supersonic	**bc_inflow_supersonic** **bc_Dirichlet** *usage rule:* choose either; **bc_inflow_supersonic** takes precedence.
bc_outflow_supersonic	**bc_outflow_supersonic** **bc_Extrapolate** *usage rule:* choose either; **bc_outflow_supersonic** takes precedence.
all others	self-matching

for a porous wall is an example), then it should somehow be reclassified. One option is to define a new **bc_type_compound** identifier and provide associated **bc_type_simple** types and a usage rule. Another option may be to allow some identifiers to be both **bc_type_simple** and **bc_type_compound** and let their appropriate use be based on context. This is still evolving.

For a given simple boundary-condition (i.e., one that is not dependent on flow conditions), the database provides a set of boundary-condition equations to be enforced through the definitions of **bc_data_set** and **bc_data**. Apart from the boundary-condition type, the precise equations to be enforced are described by boundary-condition solution data. These specified solution data are arranged by 'equation type':

Dirichlet: $Q = (Q)_{\text{specified}}$

Neumann: $\partial Q/\partial n = (\partial Q/\partial n)_{\text{specified}}$

The **Dirichlet_data** and **Neumann_data** attributes (of type **bc_data**) list both the solution variables involved in the variables (through the data-name conventions of clause G.3.3.12) and the specified solution data.

Two issues need to be addressed for specifying Dirichlet or Neumann boundary-condition data. The first is whether the data is global or local.

NOTE 2

global BC data: data applied globally to the BC patch; for example, specifying a uniform total pressure at an inflow boundary.

local BC data: data applied at each grid point of the BC patch; an example of this is varying total pressure specified at each vertex of the BC patch.

The second issue is describing the actual solution quantities that are to be specified. Both of these issues are addressed by use of the **data_array** structure.

For some types of boundary-conditions, many different combinations of solution quantities could be specified. For example, **bc_inflow_subsonic** requires four solution quantities to be specified in 3-D, but what those four quantities are varies with applications (e.g., internal versus external flows) and codes. The actual data being specified for any **bc_type** is given by the list of **data_array** instances included in the **Dirichlet_data** and **Neumann_data** attributes (actually by the identifier attached to each instance of **data_array**). This reduces the potential problem of having to specify *many* versions of a given **bc_type** (e.g., **bc_inflow_subsonic1**, **bc_inflow_subsonic2** etc.), where each has a precisely defined set of Dirichlet data. Instead, only the number of Dirichlet or Neumann quantities must be provided for each **bc_type**.

The global versus local issue can easily be handled by storing a scalar for the global BC data case, and storing an array for the local BC data case.

By convention, if the **Dirichlet_data** and **Neumann_data** are not present in an instance of **bc_data_set**, then application codes (e.g., flow solvers) are free to enforce appropriate boundary-conditions for the given type of **bc_type_simple**. Furthermore, if insufficient data is present

(e.g., only one Dirichlet variable is present for a subsonic inflow condition), then application codes are free to fill out the boundary-condition data as appropriate for the **bc_type_simple** identifier.

To facilitate implementation of boundary-conditions into existing flow solvers, if no boundary-condition data is specified, then flow solvers are free to enforce any appropriate boundary-condition equations. This includes situations where instances of **bc_data_set**, **bc_data** or **data_array** are absent within the boundary-condition hierarchy. In this case the **reference_state** specifies the reference-state conditions from which the flow solver should extract the boundary-condition data. Within the boundary-condition hierarchy, **reference_state** instances may be present at any of the **zone_bc**, **bc** or **bc_data_set** levels with the lowest taking precedence.

EXPRESS specification:

```
*)
ENTITY bc_data_set
  SUBTYPE OF (condition);
  simple_type    : bc_type_simple;
  gridloc        : OPTIONAL grid_location;
  Dirichlet_data : OPTIONAL bc_data;
  Neumann_data   : OPTIONAL bc_data;
  rstate         : OPTIONAL reference_state;
  dclass         : OPTIONAL data_class;
  dimunits       : OPTIONAL dimensional_units;
DERIVE
  location                : grid_location := NVL(gridloc, vertex);
  refstate                : reference_state := NVL(rstate, bc.refstate);
  class                   : data_class := NVL(dclass, bc.class);
  units                   : dimensional_units := NVL(dimunits, bc.units);
  vertex_list_length      : INTEGER := bc.vertex_list_length;
  face_center_list_length : INTEGER := bc.face_center_list_length;
  data_array_length       : INTEGER := bcdataset_data_array_length(SELF);
INVERSE
  bc : bc FOR data_sets;
END_ENTITY;
(*
```

Attribute definitions:

simple_type: is the boundary-condition type, which gives general information on the boundary-condition equations to be enforced.

gridloc: is non-default location information;

location: is the location of local data arrays (if any) provided in **Dirichlet_data** and **Neumann_data**. Local boundary-condition data may be defined either at vertices or boundary face

centers.

Dirichlet_data: is boundary-condition data for Dirichlet conditions which may be constant over the BC patch or defined locally at each point of the patch.

Neumann_data: is boundary-condition data for Neumann conditions which may be constant over the BC patch or defined locally at each point of the patch.

rstate: non-default reference data;

dclass: non-default data class;

dimunits: non-default dimensional units;

refstate: is reference quantities applicable to the set of boundary-condition data.

class: is the class of data contained in the boundary-condition data.

units: is the system of units employed.

vertex_list_length: is the number of vertices in the BC patch.

face_center_list_length: is the number of cell faces in the BC patch.

data_array_length: is the length of the data arrays.

bc: is the calling **bc**.

G.5.4.6 bc_data

bc_data contains a list of variables and associated data for boundary-condition specification. Each variable may be given as global data (i.e., a scalar) or local data defined at each grid point of the BC patch. By convention all data specified in a given instance of **bc_data** is to be used in the same *type* of boundary-condition equation.

EXAMPLE 1 The Dirichlet and Neumann conditions in **bc_data_set** use seperate **bc_data** structures.

This structure allows a given instance of **bc_data** to have a mixture of global and local data.

EXAMPLE 2 If the Dirichlet condition consists of a uniform stagnation pressure but with with a non-uniform velocity profile, then the stagnation pressure can be described by a scalar in the **data_global** list and the velocity by an array in the **data_local** list.

EXPRESS specification:

```
*)
ENTITY bc_data
  SUBTYPE OF (condition);
  data_global : OPTIONAL LIST OF data_array;
  data_local  : OPTIONAL LIST OF data_array;
  dclass      : OPTIONAL data_class;
```



```
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  class             : data_class := bcdata_class(SELF);
  units             : dimensional_units := bcdata_dimunits(SELF);
  data_array_length : INTEGER := NVL(Dirichlet[1].data_array_length, 0) +
                                 NVL(Neumann[1].data_array_length, 0);
INVERSE
  Dirichlet : BAG [0:1] OF bc_data_set FOR Dirichlet_data;
  Neumann   : BAG [0:1] OF bc_data_set FOR Neumann_data;
END_ENTITY;
(*
```

Attribute definitions:

data_global: is global data;

data_local: is local data;

dclass: non-default data class;

dimunits: non-default system of units;

class: is the class of data;

units: is the system of units for the data;

data_array_length: is the length data arrays;

Dirichlet: is the calling **bc_data_set** which uses this **bc_data** for Dirichlet data;

Neumann: is the calling **bc_data_set** which uses this **bc_data** for Neumann data;

Informal propositions:

ip1: An instance of **bc_data** shall be called in the role of either **Dirichlet_data** or **Neumann_data**.

G.5.4.7 reference_state

reference_state describes a reference state, which is a list of geometric or flow-state quantities defined at a common location or condition. Examples of typical reference states associated with CFD calculations are freestream, plenum, stagntation, inlet and exit.

The data that may be associated with **reference_state** is listed in Table G.3.

EXPRESS specification:

```
*)
```


Table G.3 – Data that may be associated with reference_state

data_name	Description	Units
Mach	Mach number, $M = q/c$	—
Mach_velocity	Velocity scale, q	**L/T**
Mach_velocity_sound	Speed of sound scale, c	**L/T**
Reynolds	Reynolds number, $Re = VL_R/\nu$	—
Reynods_velocity	Velocity scale, V	**L/T**
Reynolds_length	Length scale, L_R	**L**
Reynolds_viscosity_kinematic	Kinematic viscosity scale, ν	$\mathbf{L^2/T}$
length_reference	Reference length, L	**L**

```
ENTITY reference_state
  SUBTYPE OF (condition);
  state_description : OPTIONAL STRING;
  data              : LIST OF data_array;
  dclass            : OPTIONAL data_class;
  dimunits          : OPTIONAL dimensional_units;
DERIVE
  class        : data_class :=
    NVL(dclass,
        inherited_class_for_refstate(data_set, bc, zone_bc, zone, base));
  units : dimensional_units :=
    NVL(dimunits,
        inherited_units_for_refstate(data_set, bc, zone_bc, zone, base));
INVERSE
  base     : BAG [0:1] OF cfd_case FOR refstate;
  zone     : BAG [0:1] OF zone FOR refstate;
  data_set : BAG [0:1] OF bc_data_set FOR refstate;
  bc       : BAG [0:1] OF bc FOR refstate;
  zone_bc  : BAG [0:1] OF zone_bc FOR refstate;
END_ENTITY;
(*
```

Attribute definitions:

state_description: is a description of the reference state;

data: is the reference state data;

dclass: non-default data class;

dimunits: non-default dimensional units;

class: is the class of data;

units: is the system of units;

Informal propositions:

ip1: A reference state shall be called by one and only one of: **cfd_case**, **zone**, **bc_data_set**, **bc**, **zone_bc**;

ip2: The data arrays shall be consistent.

G.5.4.8 integral_data

integral_data provides a description of generic global or integral data that may be associated with a particular zone or an entire database. In contrast to **discrete_data**, integral data is not associated with any specific field location.

EXPRESS specification:

```
*)
ENTITY integral_data
  SUBTYPE OF (condition);
  data        : LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  class        : data_class :=
                      NVL(dclass, inherit_class_from_base_zone(base, zone));
  units : dimensional_units :=
                      NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_case FOR miscellaneous;
  zone : BAG [0:1] OF zone FOR global_data;
END_ENTITY;
(*
```

Attribute definitions:

data: is the data;

dclass: non-default class of data;

dimunits: non-default system of units;

class: is the class of data;

units: is the system of units;

base: is the calling **cfd_case**.

zone: is the calling **zone**.

Informal propositions:

ip1: An instance of **integral_data** shall be referenced by either a **cfd_case** or a **zone**;

ip2: The data arrays shall be consistent.

G.5.5 conditions function definitions

G.5.5.1 bcdataset_data_array_length

bcdataset_data_array_length takes a **bc_data_set** as its single argument and returns the length of the local data arrays.

EXPRESS specification:

```
*)
FUNCTION bcdataset_data_array_length(arg : bc_data_set) : INTEGER;
LOCAL
  result : INTEGER := 0;
END_LOCAL;
CASE arg.GridLocation OF
  Vertex      : result := arg.VertexListLength;
  FaceCenter,
  IFaceCenter,
  JFaceCenter,
  KFaceCenter : result := arg.face\_center\_list\_length;
END_CASE;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **bc_data_set**.

RETURNS: If the **GridLocation** is **Vertex** or **FaceCenter** or **IFaceCenter** or **JFaceCenter** or **KFaceCenter** the length of the local data arays are returned, otherwise zero is returned.

G.5.5.2 bcdata_class

bcdata_class takes a **bc_data** as its single argument and returns the class of data.

EXPRESS specification:

```
*)
FUNCTION bcdata_class(arg : bc_data) : data_class;
LOCAL
  result : data_class := ?;
END_LOCAL;
IF (EXISTS(arg.dclass)) THEN
  result := arg.dclass;
ELSE
  IF (EXISTS(arg.Dirichlet)) THEN
    result := arg.Dirichlet[1].class;
  ELSE
    IF (EXISTS(arg.Neumman)) THEN
      result := arg.Neumman[1].class;
    END_IF;
  END_IF;
END_IF;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **bc_data_set**.

RETURNS: The **data_class** of the **bc_data_set**.

G.5.5.3 bcdata_dimunits

bcdata_dimunits takes a **bc_data** as its single argument and returns the dimensional units.

EXPRESS specification:

```
*)
FUNCTION bcdata_dimunits(arg : bc_data) : dimensional_units;
LOCAL
  result : dimensional_units := ?;
END_LOCAL;
IF (EXISTS(arg.dimunits)) THEN
  result := arg.dimunits;
ELSE
  IF (EXISTS(arg.Dirichlet)) THEN
    result := arg.Dirichlet[1].units;
  ELSE
    IF (EXISTS(arg.Neumann)) THEN
      result := arg.Neumann[1].units;
```



```
    END_IF;
  END_IF;
END_IF;
RETURN(result);
END_FUNCTION;
(*
```

Argument definitions:

arg: A **bc_data_set**.

RETURNS: The **units** of the **bc_data_set**.

G.5.5.4 inherit_class_from_base_zone

Given either a **cfd_case** or a **zone** the function returns the **class**.

EXPRESS specification:

```
*)
FUNCTION inherit_class_from_base_zone(base : AGGREGATE OF cfd_case;
                                      zone : AGGREGATE OF zone) : data_class;
  IF (SIZEOF(base) > 0) THEN
    RETURN(base[1].class);
  ELSE
    IF (SIZEOF(zone) > 0) THEN
      RETURN(zone[1].class);
    END_IF;
  END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

base: A possibly empty aggregate of **cfd_case**.

zone: A possibly empty aggregate of **zone**

RETURNS: The value of the **class** attribute for the first **cfd_case** of the first **zone**. If there is an error in the arguments, indeterminate is returned.

G.5.5.5 inherit_units_from_base_zone

Given either a **cfd_case** or a **zone** the function returns the **units**.

EXPRESS specification:

```
*)
FUNCTION inherit_units_from_base_zone(base : AGGREGATE OF cfd_case;
                                      zone : AGGREGATE OF zone) : dimensional_units;
  IF (SIZEOF(base) > 0) THEN
    RETURN(base[1].units);
  ELSE
    IF (SIZEOF(zone) > 0) THEN
      RETURN(zone[1].units);
    END_IF;
  END_IF;
  RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

base: A possibly empty aggregate of **cfd_case**.

zone: A possibly empty aggregate of **zone**

RETURNS: The value of the **units** attribute for the first **cfd_case** of the first **zone**. If there is an error in the arguments, indeterminate is returned.

G.5.5.6 inherited_class_for_refstate

The **inherited_class_for_refstate** function determines a class of data.

EXPRESS specification:

```
*)
FUNCTION inherited_class_for_refstate(bd : AGGREGATE OF bc_data_set;
                                      bc : AGGREGATE OF bc;
                                      zb : AGGREGATE OF zone_bc;
                                      zn : AGGREGATE OF zone;
                                      db : AGGREGATE OF cfd_case) : data_class;
IF (SIZEOF(bd) > 0) THEN RETURN(bd[1].class); END_IF;
IF (SIZEOF(bc) > 0) THEN RETURN(bc[1].class); END_IF;
IF (SIZEOF(zb) > 0) THEN RETURN(zb[1].class); END_IF;
IF (SIZEOF(zn) > 0) THEN RETURN(zn[1].class); END_IF;
IF (SIZEOF(db) > 0) THEN RETURN(db[1].class); END_IF;
RETURN(?);
END_FUNCTION;
(*
```


Argument definitions:

bd: A possibly empty aggregate of **bc_data_set**;

bc: A possibly empty aggregate of **bc**;

zb: A possibly empty aggregate of **zone_bc**;

zn: A possibly empty aggregate of **zone**;

db: A possibly empty aggregate of **cfd_case**;

RETURNS: The value of the **data_class** which has the highest precedence. If there is an error in the arguments, indeterminate is returned.

G.5.5.7 inherited_units_for_refstate

The **inherited_units_for_refstate** function determines dimensional units.

EXPRESS specification:

```
*)
FUNCTION inherited_units_for_refstate(bd : AGGREGATE OF bc_data_set;
                                      bc : AGGREGATE OF bc;
                                      zb : AGGREGATE OF zone_bc;
                                      zn : AGGREGATE OF zone;
                                      db : AGGREGATE OF cfd_case) : dimensional_units;
IF (SIZEOF(bd) > 0) THEN RETURN(bd[1].units); END_IF;
IF (SIZEOF(bc) > 0) THEN RETURN(bc[1].units); END_IF;
IF (SIZEOF(zb) > 0) THEN RETURN(zb[1].units); END_IF;
IF (SIZEOF(zn) > 0) THEN RETURN(zn[1].units); END_IF;
IF (SIZEOF(db) > 0) THEN RETURN(db[1].units); END_IF;
RETURN(?);
END_FUNCTION;
(*
```

Argument definitions:

bd: A possibly empty aggregate of **bc_data_set**;

bc: A possibly empty aggregate of **bc**;

zb: A possibly empty aggregate of **zone_bc**;

zn: A possibly empty aggregate of **zone**;

db: A possibly empty aggregate of **cfd_case**;

RETURNS: The value of the **dimensional_units** which has the highest precedence. If there is an error in the arguments, indeterminate is returned.

EXPRESS specification:

```
*)
END_SCHEMA;  -- end of conditions
(*
```

G.6 equations

The following EXPRESS declaration begins the **equations** schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA equations;
  REFERENCE FROM basis;
  REFERENCE FROM hierarchy;
(*
```

G.6.1 Introduction

This schema defines and describes governing flow-equation set associated with Navier-Stokes CFD codes.

The graphical form for the **equations** schema is given in Figures G.28 through G.37.

G.6.2 Fundamental concepts and assumptions

The description of the flow-equation set includes the general class of governing equations, the turbulent closure equations, the gas model, and the viscosity and thermal-conductivity models. Included with each equation description are associated constants.

The structure definitions attempt to balance the opposing requirements of initial ease of implementation against requirements for extensibilty and future growth.

The intended use of these structures is primarily for archival purposes and to provide additional documentation of the flow solution.

G.6.3 equations type definitions

G.6.3.1 turbulence_data_name

turbulence_data_name is an enumeration of standardized Reynolds-averaged Navier-Stokes turbulence model variables.

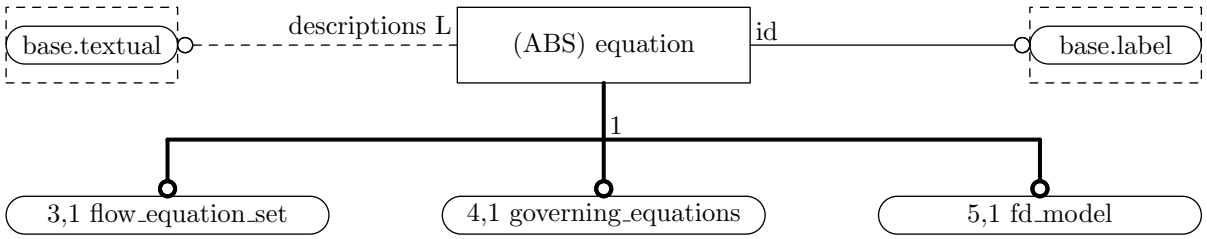


Figure G.28 – Entity level diagram of ARM equations schema (page 1 of 10)

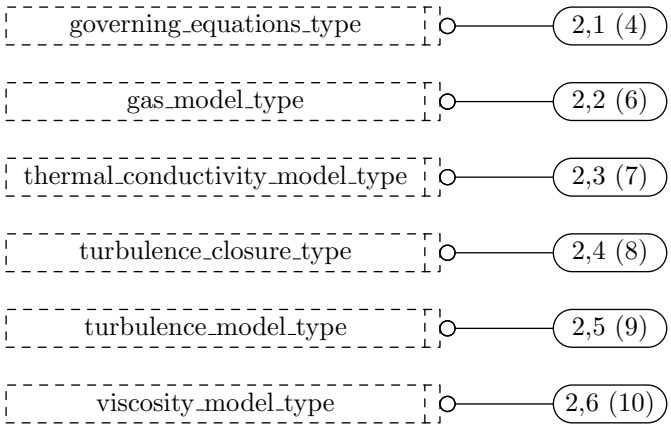


Figure G.29 – Entity level diagram of ARM equations schema (page 2 of 10)

Turbulence model solution quantities and model constants present a particularly difficult nomenclature problem — to be precise it is necessary to identify both the variable and the model (and version) that it comes from. The list (in Table 7) falls short in this respect.

EXPRESS specification:

```
*)
TYPE turbulence_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

The required identifiers and their meanings are given in Table 7.

G.6.3.2 force_moment_data_name

force_moment_data_name is an enumeration of standardized force and moment data.

Conventions for data-name identifiers for forces and moments are defined in this subclause.

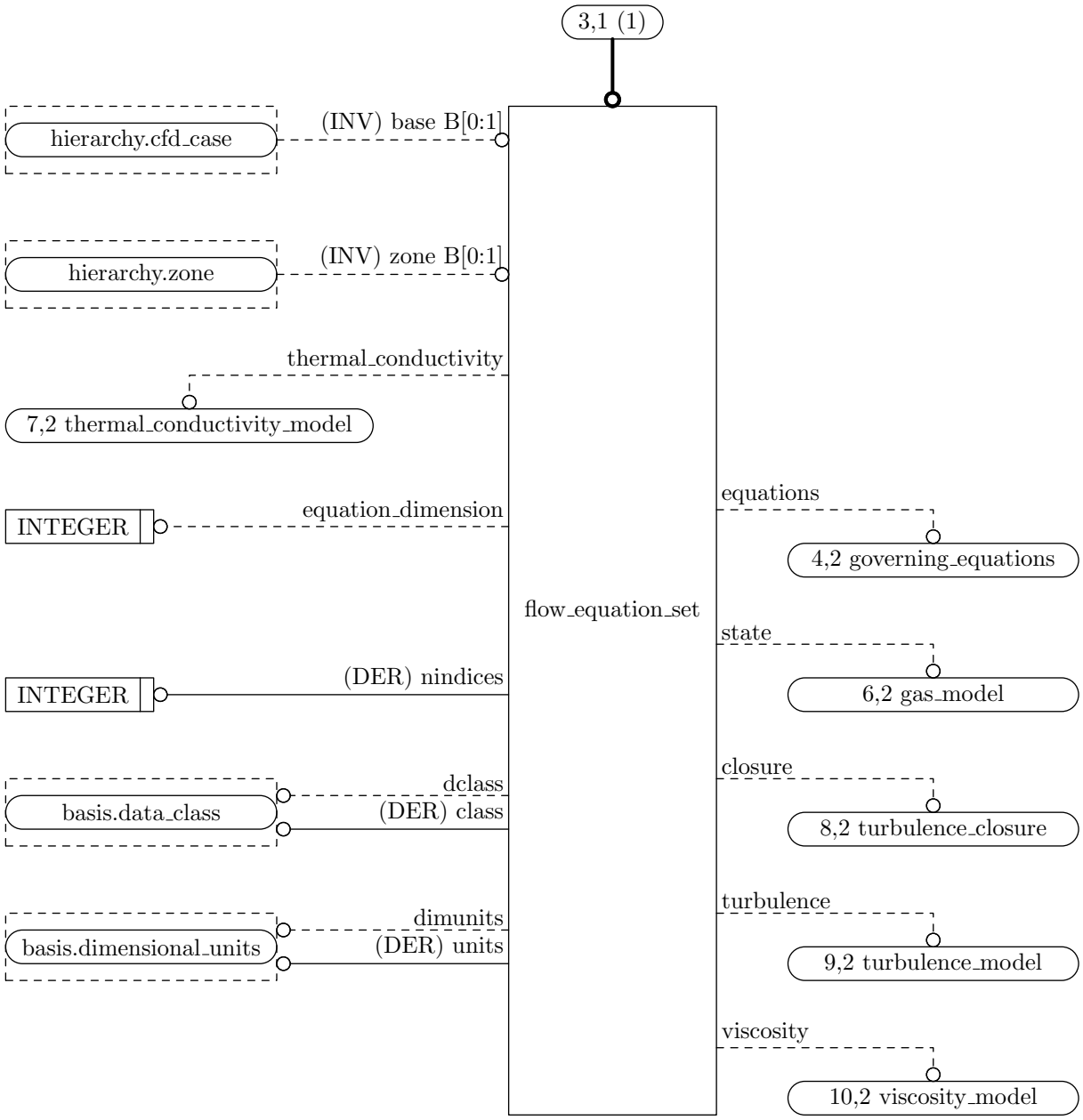


Figure G.30 – Entity level diagram of ARM equations schema (page 3 of 10)

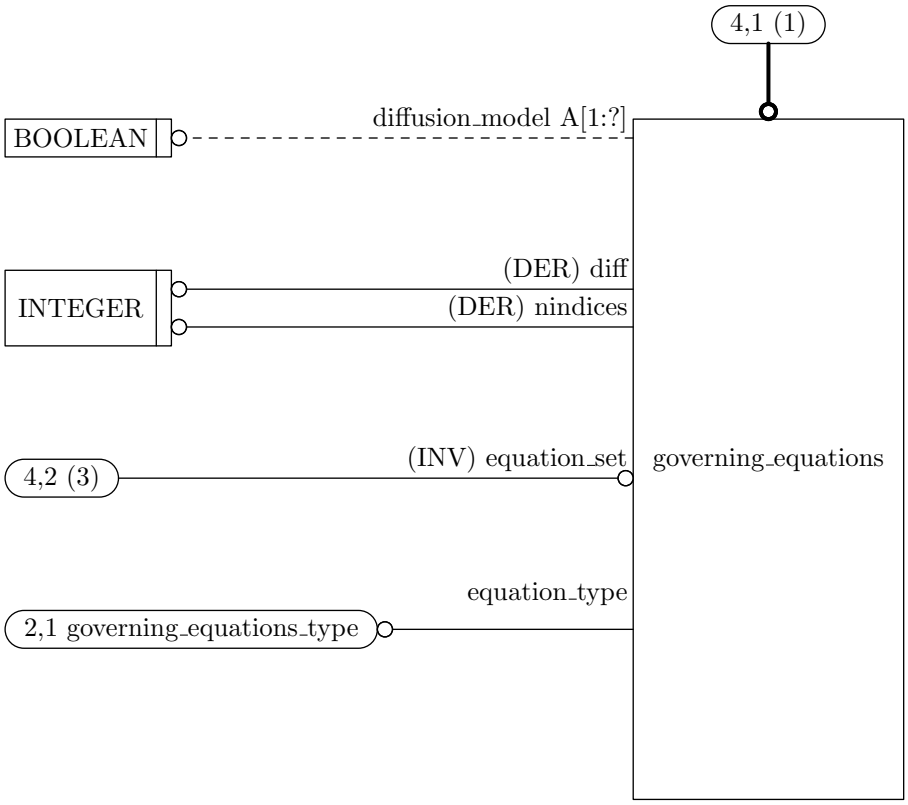


Figure G.31 – Entity level diagram of ARM equations schema (page 4 of 10)

Given a differential force $\vec{f}$ (i.e. a force per unit area), the force integrated over a surface is,

$$\vec{F} = F_x\hat{e}_x + F_y\hat{e}_y + F_z\hat{e}_z = \int \vec{f}\, dA,$$

where $\hat{e}_x$, $\hat{e}_y$ and $\hat{e}_z$ are the unit vectors in the x, y and z directions, respectively. The moment about a point $\vec{r}_0$ integrated over a surface is,

$$\vec{M} = M_x\hat{e}_x + M_y\hat{e}_y + M_z\hat{e}_z = \int (\vec{r} - \vec{r}_0) \times \vec{f}\, dA.$$

Lift and drag components of the integrated force are,

$$L = \vec{F} \cdot \hat{L} \qquad D = \vec{F} \cdot \hat{D}$$

where $\hat{L}$ and $\hat{D}$ are the unit vectors in the positive lift and drag directions, respectively.

Lift, drag and moment are often computed in auxiliary coordinate frames (e.g. wind axes or stability axes). We introduce the convention that lift, drag and moment are computed in the (ξ, η, ζ) coordinate system. Positive drag is assumed parallel to the ξ-direction (i.e. $\hat{D} = \hat{e}_\xi$); and positive lift is assumed parallel to the η-direction (i.e. $\hat{L} = \hat{e}_\eta$). Thus, forces and moments defined in this auxiliary coordinate system are,

$$L = \vec{F} \cdot \hat{e}_\eta \qquad D = \vec{F} \cdot \hat{e}_\xi$$

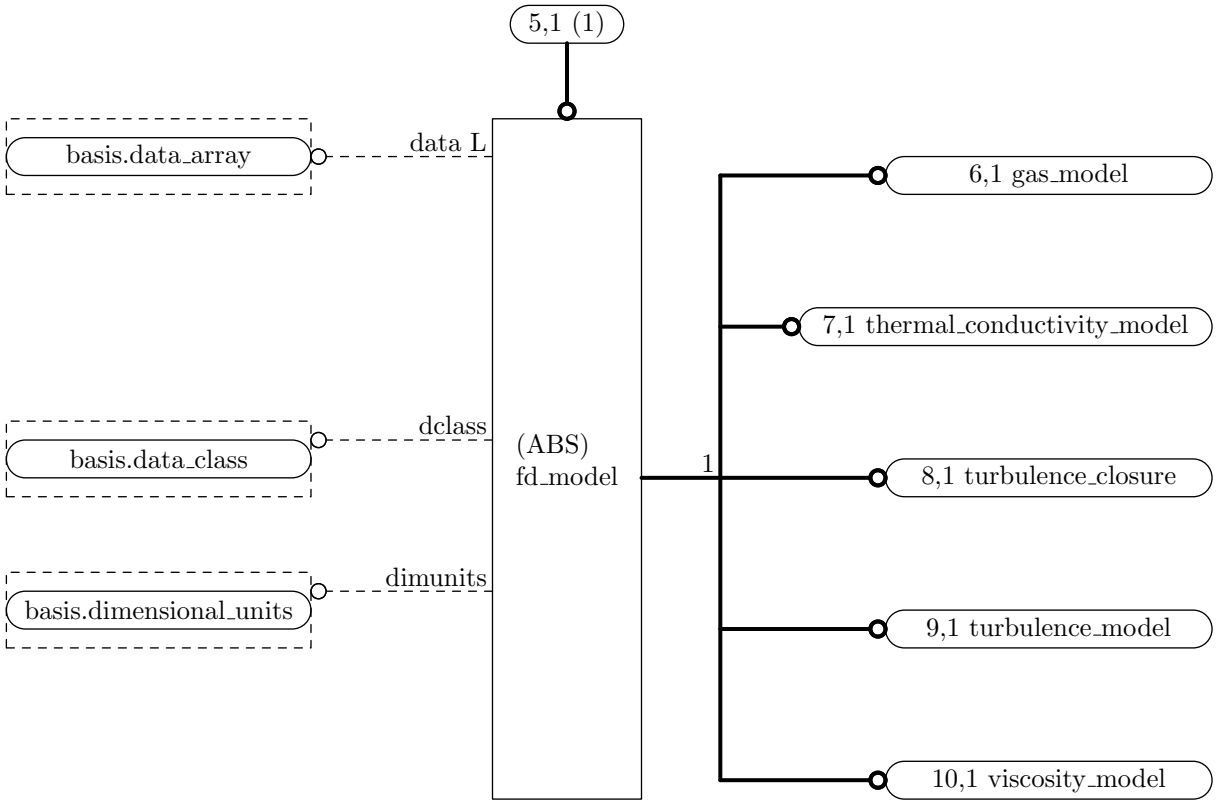


Figure G.32 – Entity level diagram of ARM equations schema (page 5 of 10)

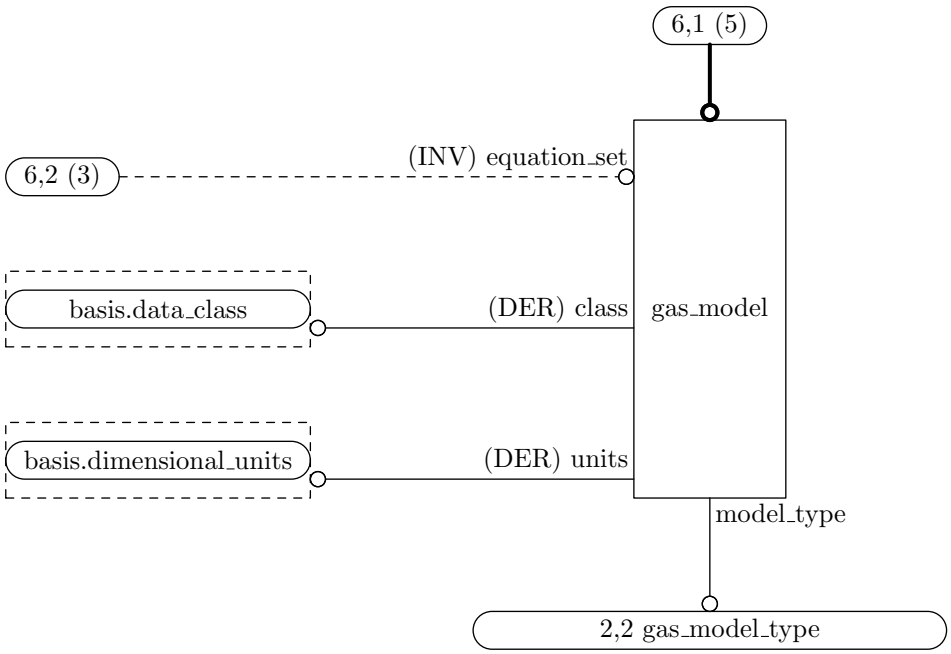


Figure G.33 – Entity level diagram of ARM equations schema (page 6 of 10)

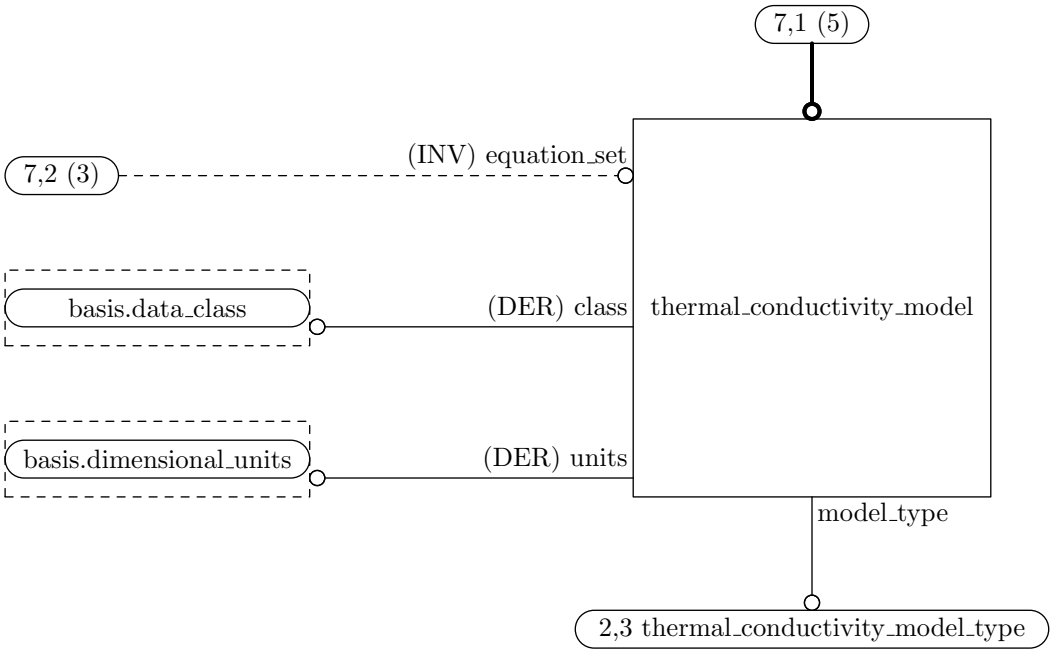


Figure G.34 – Entity level diagram of ARM equations schema (page 7 of 10)

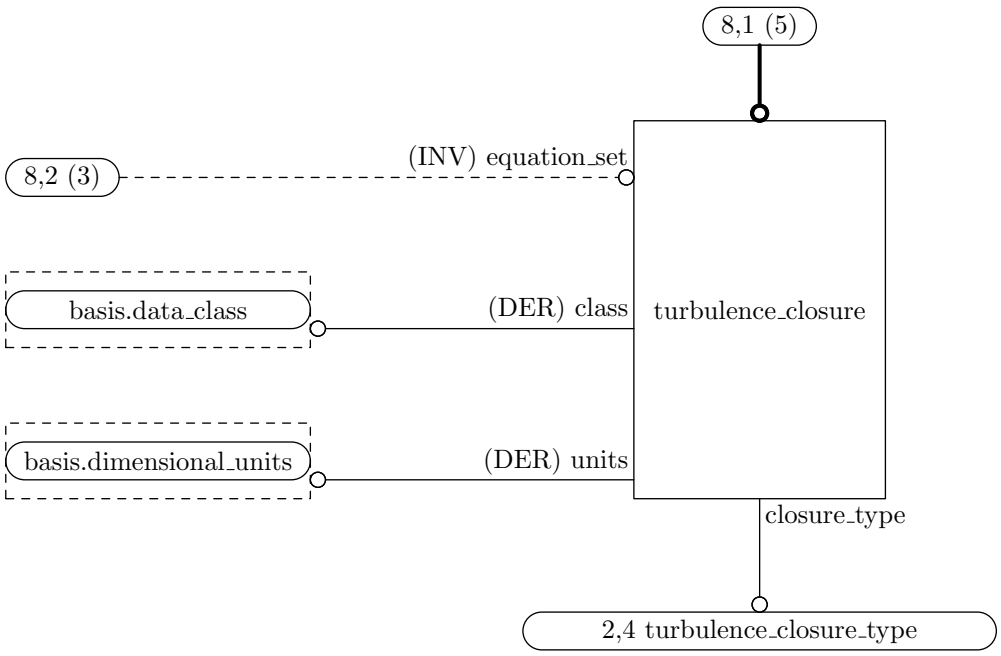


Figure G.35 – Entity level diagram of ARM equations schema (page 8 of 10)

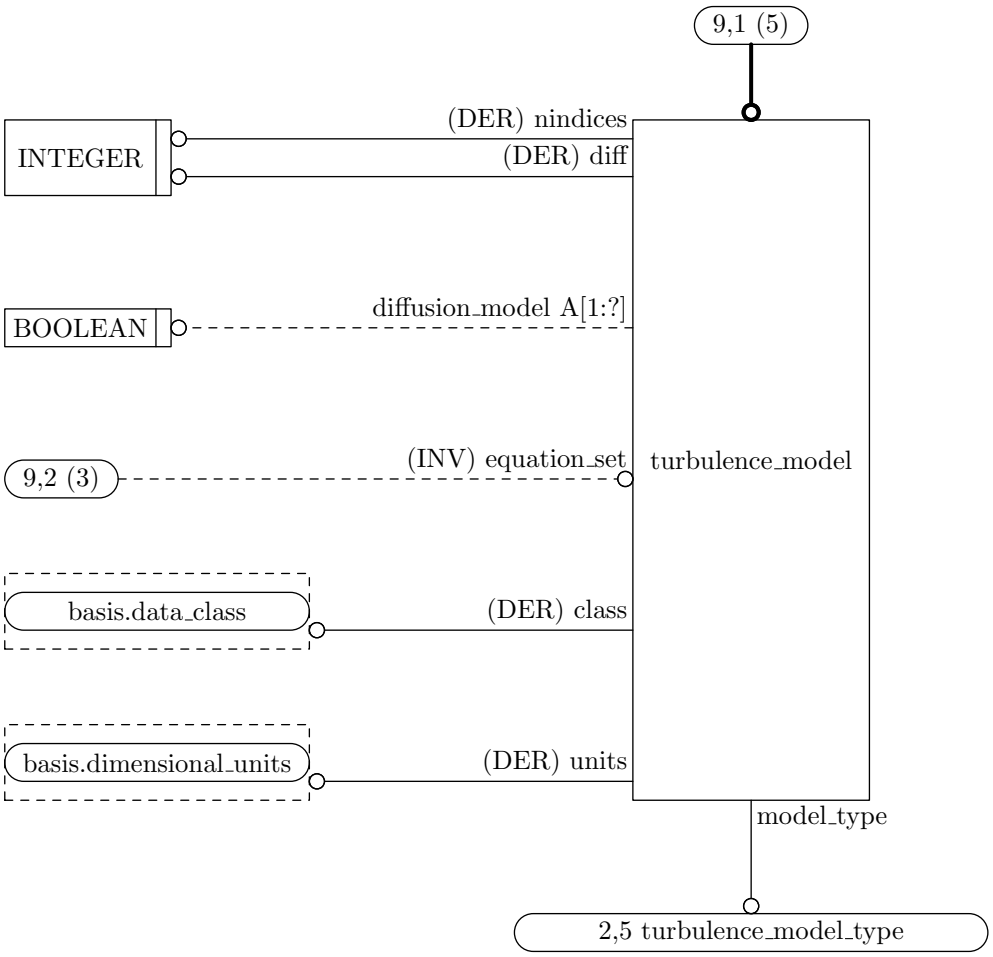


Figure G.36 – Entity level diagram of ARM equations schema (page 9 of 10)

$$\vec{M} = M_\xi \hat{e}_\xi + M_\eta \hat{e}_\eta + M_\zeta \hat{e}_\zeta = \int (\vec{r} - \vec{r}_0) \times \vec{f}\, dA.$$

Lift, drag and moment coefficients in 3–D are defined as,

$$C_L = \frac{L}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 S} \qquad C_D = \frac{D}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 S} \qquad \vec{C}_M = \frac{\vec{M}}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 c_{\text{ref}} S_{\text{ref}}},$$

where $\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2$ is a reference dynamic pressure, S_{ref} is a reference area, and c_{ref} is a reference length. For a wing, S_{ref} is typically the wing area and c_{ref} is the mean aerodynamic chord. In 2–D, the sectional force coefficients are,

$$c_l = \frac{L'}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 c_{\text{ref}}} \qquad c_d = \frac{D'}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 c_{\text{ref}}} \qquad \vec{c}_m = \frac{\vec{M}'}{\frac{1}{2}\rho_{\text{ref}} q_{\text{ref}}^2 c_{\text{ref}}^2},$$

where the forces are integrated along a contour (e.g. an airfoil cross-section) rather than a surface.

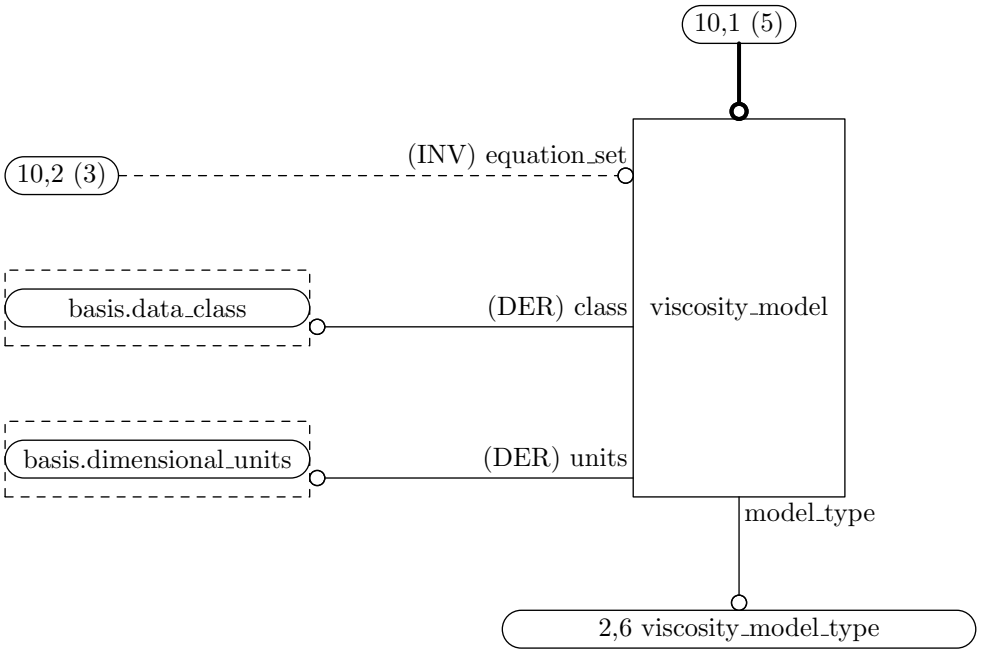


Figure G.37 – Entity level diagram of ARM equations schema (page 10 of 10)

The following data-name identifiers and definitions are provided for forces and moments and their associated coefficients. For coefficients, the dynamic pressure and length scales used in the normalization are provided.

EXPRESS specification:

```
*)
TYPE force_moment_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

The required identifiers and their meanings are given in Table 10.

G.6.3.3 governing_equations_type

governing_equations_type is an enumeration of the classes of flow equations.

EXPRESS specification:

```
*)
TYPE governing_equations_type = ENUMERATION OF
     (unspecified,
```



```
      user_defined,
      full_potential,
      Euler,
      NS_laminar,
      NS_turbulent,
      NS_laminar_incompressible,
      NS_turbulent_incompressible);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

user_defined: is specified via an external agreement between the data creator and the data user;

full_potential: is full potential flow;

Euler: is Euler flow;

NS_laminar: is Navier-Stokes laminar flow;

NS_turbulent: is Navier-Stokes turbulent flow;

NS_laminar_incompressible: is Navier-Stokes laminar incrompressible flow;

NS_turbulent_incompressible: is Navier-Stokes turbulent incrompressible flow;

G.6.3.4 gas_model_type

gas_model_type is an enumeration of the state models relating pressure, temperature and density.

EXPRESS specification:

```
*)
TYPE gas_model_type = ENUMERATION OF
     (unspecified,
      user_defined,
      ideal,
      Van_der_Waals);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified.

user_defined: is specified via an external agreement between the data creator and the data user;

ideal: the state model is the perfect gas law. The pressure, temperature and density are related by,

$$p = \rho R T,$$

where R is the ideal gas constant. Related quantities are the specific heat at constant pressure (c_p), specific heat at constant volume (c_v) and specific heat ratio ($\gamma = c_p/c_v$). The gas constant and specific heats are related by $R = c_p - c_v$.

The **standard_data_name** identifiers associated with the perfect gas law are: **ideal_gas_constant**, **specific_heat_ratio**, **specific_heat_volume** and **specific_heat_pressure**. These are described in clause G.3.3.12.

Van_der_Waals: the state model is Van der Waals' equation.

G.6.3.5 viscosity_model_type

viscosity_model_type is an enumeration of the relationships between molecular viscosity and temperature.

EXPRESS specification:

```
*)
TYPE viscosity_model_type = ENUMERATION OF
     (unspecified,
      user_defined,
      constant_viscosity,
      power_law,
      Sutherland_law);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

user_defined: is specified via an external agreement between the data creator and the data user;

constant_viscosity: the molecular viscosity is constant throughout the field and is equal to some reference value ($\mu = \mu_{\text{ref}}$)

power_law: the molecular viscosity follows a power-law relation,

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^n .$$

The **standard_data_name** identifiers associated with this model are: **power_law_exponent**, **temperature_reference** and **viscosity_molecular_reference**. These are described in clause G.3.3.12.

Sutherland_law: Sutherland's Law for molecular viscosity,

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{3/2} \frac{T_{\text{ref}} + T_s}{T + T_s},$$

where T_s is the Sutherland Law constant, and μ_{ref} and T_{ref} are the reference viscosity and temperature, respectively.

The **standard_data_name** identifiers associated with this model are: **Sutherland_law_constant**, **temperature_reference** and **viscosity_molecular_reference**. These are described in clause G.3.3.12.

NOTE 1

For air [4], the power-law exponent is $n = 0.666$, Sutherlands Law constant (T_s) is 110.6 K, the reference temperature (T_{ref}) is 273.15 K, and the reference viscosity (μ_{ref}) is 1.716×10^{-5} kg/(m s).

G.6.3.6 thermal_conductivity_model_type

thermal_conductivity_model_type is an enumeration of the relationships between the thermal-conductivity coefficient and temperature.

EXPRESS specification:

```
*)
TYPE thermal_conductivity_model_type = ENUMERATION OF
     (unspecified,
      user_defined,
      constant_Prandtl,
      power_law,
      Sutherland_law);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

user_defined: is specified via an external agreement between the data creator and the data user;

constant_Prandtl: the Prandtl number ($Pr = \mu c_p / k$) is constant and equal to some reference value.

The **standard_data_name** identifier associated with this model is **constant_Prandtl**, and is described in clause G.3.3.12.

power_law: the thermal conductivity is related to temperature via a power-law.

$$k = k_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^n .$$

The **standard_data_name** identifiers associated with this model are: **power_law_exponent**, **temperature_reference** and **thermal_conductivity_reference**. These are described in clause G.3.3.12.

Sutherland_law: Sutherland's Law for thermal conductivity.

$$k = k_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{3/2} \frac{T_{\text{ref}} + T_s}{T + T_s},$$

where T_s is the Sutherland Law constant, and k_{ref} and T_{ref} are the reference thermal conductivity and temperature, respectively.

The **standard_data_name** identifiers associated with this model are: **Sutherland_law_constant**, **temperature_reference** and **thermal_conductivity_molecular_reference**. These are described in clause G.3.3.12.

NOTE 1

For air [4], the Prandtl number is $Pr = 0.72$, the power-law exponent is $n = 0.81$, Sutherlands Law constant (T_s) is 194.4 K, the reference temperature (T_{ref}) is 273.15 K, and the reference thermal conductivity (k_{ref}) is 2.414×10^{-2} $kg\ m/(s^3 K)$.

G.6.3.7 turbulence_closure_type

turbulence_closure_type is an enumeration of the kinds of turbulence closure for the Reynolds stress terms of the Navier-Stokes equations.

EXPRESS specification:

```
*)
TYPE turbulence_closure_type = ENUMERATION OF
     (unspecified,
      user_defined,
      eddy_viscosity,
      Reynolds_stress,
      Reynolds_stress_algebraic);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

user_defined: is specified via an external agreement between the data creator and the data user;

eddy_viscosity: Boussinesq eddy-velocity closure. The Reynolds stresses are approximated as the product of an eddy viscosity (ν_t) and the mean strain tensor. Using indicial notation, the relation is,

$$-\overline{u_i\prime u_j\prime} = \nu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)$$

where $-\overline{u_i\prime u_j\prime}$ are the Reynolds stresses.

Reynolds_stress: no approximation of the Reynolds stresses.

Reynolds_stress_algebraic: an algebraic approximation for the Reynolds stresses based on some intermediate transport quantities.

The associated **standard_data_name** name identifiers are: **eddy_viscosity** and **Prandtl_-turbulent**. These are described in clause G.3.3.12.

G.6.3.8 turbulence_model_type

turbulence_model_type is an enumeration of the equation sets for modeling the turbulence quantities.

EXPRESS specification:

```
*)
TYPE turbulence_model_type = ENUMERATION OF
     (unspecified,
      user_defined,
      algebraic_Baldwin_Lomax,
      algebraic_Cebeci_Smith,
      half_equation_Johnson_King,
      one_equation_Baldwin_Barth,
      one_equation_Spalart_Allmaras,
      two_equation_Jones_Launder,
      two_equation_Menter_SST,
      two_equation_Wilcox);
END_TYPE;
(*
```

Enumerated item definitions:

unspecified: is unspecified;

user_defined: is specified via an external agreement between the data creator and the data user;

algebraic_Baldwin_Lomax: is Baldwin-Lomax;

algebraic_Cebeci_Smith: is Cebeci-Smith;

half_equation_Johnson_King: is Johnson-King;

one_equation_Baldwin_Barth: is Baldwin-Barth;

one_equation_Spalart_Allmaras: is Spalart-Allmaras;

two_equation_Jones_Launder: is Jones-Launder;

two_equation_Menter_SST: is Menter;

two_equation_Wilcox: is Wilcox.

The associated **standard_data_name** name identifiers for the Spalart-Allmaras turbulence model (version Ia) are: **turbulent_SA_cb1**, **turbulent_SA_cb2**, **turbulent_SA_sigma**, **turbulent_SA_kappa**, **turbulent_SA_cw1**, **turbulent_SA_cw2**, **turbulent_SA_cw3**, **turbulent_SA_cv1**, **turbulent_SA_ct1**, **turbulent_SA_ct2**, **turbulent_SA_ct3**, and **turbulent_SA_ct4**. These are described in clause G.3.3.12.

G.6.4 equations entity definitions

G.6.4.1 equation

A **equation** represents the concept of a mathematical formulation of a physics phenonema.

EXPRESS specification:

```
*)
ENTITY equation;
  descriptions : OPTIONAL LIST OF textual;
  id           : label;
END_ENTITY;
(*
```

Attribute definitions:

descriptions: is annotation;

id: User-specified instance identifier;

G.6.4.2 flow_equation_set

flow_equation_set is a general description of governing flow equations. It includes the dimensionality of the governing equations.

EXPRESS specification:

```
*)
ENTITY flow_equation_set
  SUBTYPE OF (equation);
  dimension            : OPTIONAL INTEGER;
  equations            : OPTIONAL governing_equations;
  state                : OPTIONAL gas_model;
  viscosity            : OPTIONAL viscosity_model;
  thermal_conductivity : OPTIONAL thermal_conductivity_model;
  closure              : OPTIONAL turbulence_closure;
  turbulence           : OPTIONAL turbulence_model;
  dclass               : OPTIONAL data_class;
  dimunits             : OPTIONAL dimensional_units;
DERIVE
  nindices : INTEGER := NVL(base[1].nindices, 0) +
                        NVL(zone[1].nindices, 0);
  class     : data_class :=
                    NVL(dclass, inherit_class_from_base_zone(base, zone));
  units     : dimensional_units :=
                    NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_case FOR equations;
  zone : BAG [0:1] OF zone FOR equations;
END_ENTITY;
(*
```

Attribute definitions:

dimension: is the dimensionality of the governing equations; it is the number of spatial variables describing the flow.

equations: describes the general class of equations.

state: describes the equation of state.

viscosity: describes the auxiliary relations for molecular viscosity.

thermal_conductivity: describes the auxiliary relations for the thermal conductivity coefficient.

closure: describes the turbulent closure for Reynolds-averaged Navier-Stokes equations.

turbulence: describes the turbulence model for Reynolds-averaged Navier-Stokes equations.

dclass: non-default class of data;

dimunits: non-default system of units;

class: is the class of data contained in the **flow_equation_set**.

units: is the system of units.

nindices: The number of indices required to reference a node.

base: is the calling calling **cfd_case**;

zone: is the calling calling **zone**;

Informal propositions:

ip1: A **flow_equation_set** shall be called by either a **cfd_case** or by a **zone**.

G.6.4.3 governing_equations

governing_equations describes the class of governing flow equations associated with the solution.

EXPRESS specification:

```
*)
ENTITY governing_equations
  SUBTYPE OF (equation);
  equation_type   : governing_equations_type;
  diffusion_model : OPTIONAL ARRAY [1:diff] OF BOOLEAN;
DERIVE
  nindices : INTEGER := equation_set.nindices;
  diff     : INTEGER := (nindices**2 + nindices)/2;
INVERSE
  equation_set : flow_equation_set FOR equations;
END_ENTITY;
(*
```

Attribute definitions:

equation_type: is the kind of equation;

diffusion_model: describes the viscous diffusion terms modelled in the flow equations, and is applicable only to Navier-Stokes equations. Typically, thin-layer approximations include only the diffusion terms in one or two computational-coordinate directions. **diffusion_model** encodes the coordinate directions that include second-derivative and cross-derivative diffusion terms. The first **dimension** elements are second-derivative terms and the remainder elements are cross-derivative terms. A value of TRUE indicates the diffusion term is modelled, and FALSE indicates that it is not modelled. In 3–D, the encoding of **diffusion_model** is given in Table G.4, where derivatives in the i, j and k computational-coordinates are ξ, η and ζ, respectively.

EXAMPLE 1 The full Navier-Stokes equations in 3–D are indicated by:
diffusion_model = `[TRUE,TRUE,TRUE,TRUE,TRUE,TRUE]` while the thin-layer equations including only diffusion in the j-direction are indicated by:
diffusion_model = `[FALSE,TRUE,FALSE,FALSE,FALSE,FALSE]`.

Table G.4 – Encoding of the 3-D diffusion_model

Element	Modelled terms
$n = 1$	diffusion terms in i ($\partial^2/\partial\xi^2$)
$n = 2$	diffusion terms in j ($\partial^2/\partial\eta^2$)
$n = 3$	diffusion terms in k ($\partial^2/\partial\zeta^2$)
$n = 4$	cross-diffusion terms in i-j ($\partial^2/\partial\xi\partial\eta$ and $\partial^2/\partial\eta\partial\xi$)
$n = 5$	cross-diffusion terms in j-k ($\partial^2/\partial\eta\partial\zeta$ and $\partial^2/\partial\zeta\partial\eta$)
$n = 6$	cross-diffusion terms in k-i ($\partial^2/\partial\zeta\partial\xi$ and $\partial^2/\partial\xi\partial\zeta$)

nindices: The number of indices required to reference a node.

diff: is the number of elements in **diffusion_model**. For 1-D this is one, for 2-D it is three, and for 3-D it is six.

equation_set: is the calling **flow_equation_set**.

G.6.4.4 fd_model

EXPRESS specification:

```
*)
ENTITY fd_model
  SUBTYPE OF (equation);
  data        : OPTIONAL LIST OF data_array;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_fd_model FOR fd_model;
  ABSTRACT SUPERTYPE;
  ONEOF(gas_model,
        thermal_conductivity_model,
        turbulence_closure,
        turbulence_model,
        viscosity_model);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

data: is the data;

dclass: is non-default class of data;

dimunits: is non-default dimensional units.

Informal propositions:

ip1: The **data** shall be consistent.

G.6.4.5 gas_model

gas_model describes the equation of state model used in the governing equations to relate pressure, temperature and density.

EXPRESS specification:

```
*)
ENTITY gas_model
  SUBTYPE OF (fd_model);
  model_type : gas_model_type;
DERIVE
  class        : data_class :=
                      NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                      NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR state;
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation of state model.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

G.6.4.6 thermal_conductivity_model

thermal_conductivity_model describes the model for relating the thermal-conductivity coefficient (k) to temperature.

EXPRESS specification:

```
*)
ENTITY thermal_conductivity_model
  SUBTYPE OF (fd_model);
  model_type : thermal_conductivity_model_type;
```



```
DERIVE
  class        : data_class :=
                      NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                      NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR thermal_conductivity;
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular model type.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

G.6.4.7 turbulence_closure

turbulence_closure describes the turbulence closure for the Reynolds stress terms of the Navier-Stokes equations.

EXPRESS specification:

```
*)
ENTITY turbulence_closure
  SUBTYPE OF (fd_model);
  closure_type : turbulence_closure_type;
DERIVE
  class        : data_class :=
                      NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                      NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR closure;
END_ENTITY;
(*
```

Attribute definitions:

closure_type: is the particular closure type.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

G.6.4.8 turbulence_model

turbulence_model describes the equation set used to model the turbulence quantities.

.

EXPRESS specification:

```
*)
ENTITY turbulence_model
  SUBTYPE OF (fd_model);
  model_type      : turbulence_model_type;
  diffusion_model : OPTIONAL ARRAY [1:diff] OF BOOLEAN;
DERIVE
  class        : data_class :=
                      NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                      NVL(SELF\fd_model.dimunits, equation_set.units);
  nindices   : INTEGER := equation_set.nindices;
  diff             : INTEGER := (nindices**2 + nindices)/2;
INVERSE
  equation_set : flow_equation_set FOR turbulence;
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation set;

diffusion_model: is the description of the viscous diffusion terms included in the turbulent transport model equations;

class: is the class of data;

units: is the dimensional units;

nindices: The number of indices required to reference a node.

diff: is the number of elements in **diffusion_model**. For 1-D this is one, for 2-D it is three, and for 3-D it is six.

equation_set: is the calling **flow_equation_set**.

G.6.4.9 viscosity_model

viscosity_model describes the model for relating molecular viscosity (μ) to temperature.

EXPRESS specification:

```
*)
ENTITY viscosity_model
  SUBTYPE OF (fd_model);
  model_type : viscosity_model_type;
DERIVE
  class        : data_class :=
                      NVL(SELF\fd_model.dclass, equation_set.class);
  units : dimensional_units :=
                      NVL(SELF\fd_model.dimunits, equation_set.units);
INVERSE
  equation_set : flow_equation_set FOR viscosity;
END_ENTITY;
(*
```

Attribute definitions:

model_type: is the particular equation of state model.

class: is the class of data;

units: is the dimensional units;

equation_set: is the calling **flow_equation_set**.

EXPRESS specification:

```
*)
END_SCHEMA; -- end of equations
(*
```

G.7 results

The following EXPRESS declaration begins the **results** schema and identifies the necessary external references.

EXPRESS specification:

```
*)
SCHEMA results;
  REFERENCE FROM basis;
  REFERENCE FROM hierarchy;
(*
```

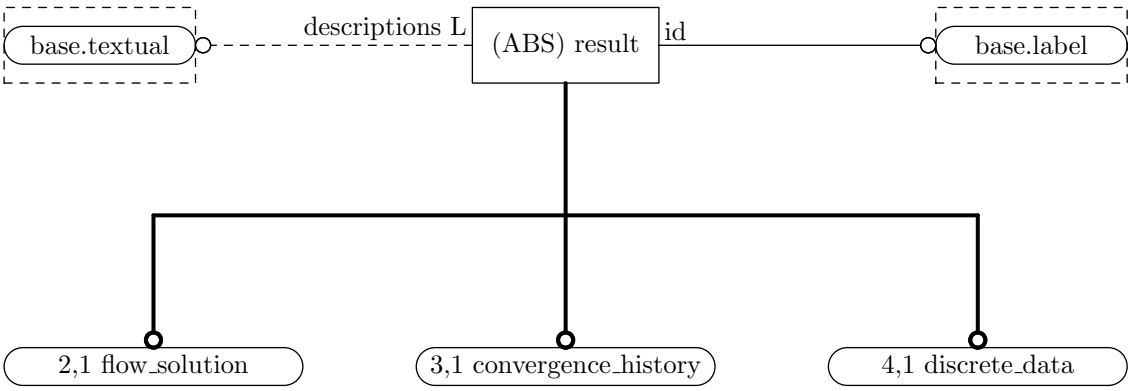



Figure G.38 – Entity level diagram of ARM results schema (page 1 of 4)

G.7.1 Introduction

This schema defines and describes flow solution data and other data resulting from a CFD analysis.

The graphical form for the **results** schema is given in Figures G.38 through G.41.

G.7.2 Fundamental concepts and assumptions

The principal result of a CFD analysis is the flow solution data over the computational grid.

Other data, such as equation residuals, can also form part of the results.

G.7.3 results type definitions

G.7.3.1 flow_solution_data_name

flow_solution_data_name is an enumeration of standardized flow solution data.

This clause describes data-name identifiers for typical Navier-Stokes solution variables. The list is obviously incomplete, but should suffice for initial implementation of the CFD system. The variables listed in this section are dimensional or raw quantities; nondimensional parameters and coefficients based on these variables are discussed in G.3.3.14.

A reasonably universal notation is used for state variables. Static quantities are measured with the fluid at speed: static density (ρ), static pressure (p), static temperature (T), static internal energy per unit mass (e), static enthalpy per unit mass (h), entropy (s), and static speed of sound (c). The true entropy isaa approximated by the function $\tilde{s} = p/\rho^{\gamma}$ (this assumes an ideal gas). The velocity is $\vec{q} = u\hat{e}_x + v\hat{e}_y + w\hat{e}_z$, with magnitude $q = \sqrt{\vec{q}\cdot\vec{q}}$. Stagnation quantities are obtained by bringing the fluid isentropically to rest; these are identified by a subscript '$_0$'. The term 'total' is also used to refer to stagnation quantities.

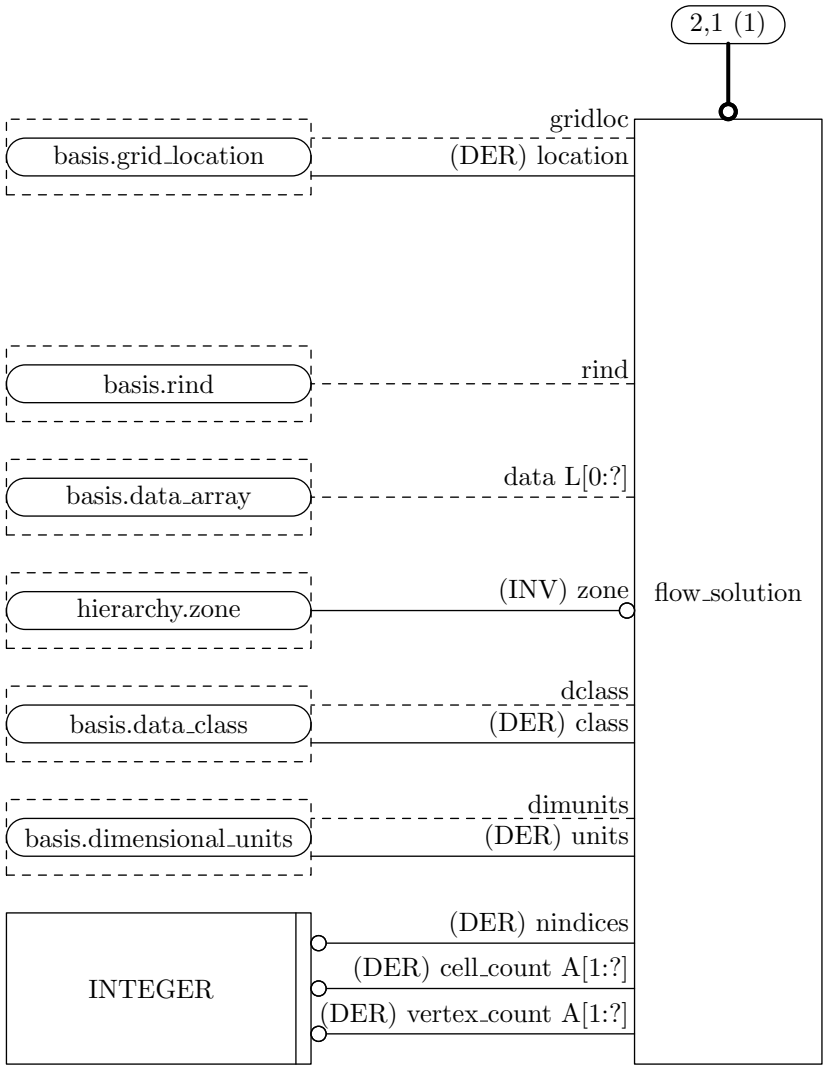


Figure G.39 – Entity level diagram of ARM results schema (page 2 of 4)

Conservation variables are density, momentum ($\rho\vec{q} = \rho u\hat{e}_x + \rho v\hat{e}_y + \rho w\hat{e}_z$), and stagnation energy per unit volume (ρe_0).

Molecular diffusion and heat transfer introduce the molecular viscosity (μ), kinematic viscosity (ν) and thermal conductivity coefficient (k). These are obtained from the state variables through auxiliary correlations. For a perfect gas, μ and k are functions of static temperature only.

The Navier-Stokes equations involve the strain tensor ($\bar{\bar{S}}$) and the shear-stress tensor ($\bar{\bar{\tau}}$). Using indicial notation, the 3–D cartesian components of the strain tensor are,

$$\bar{\bar{S}}_{i,j} = \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right),$$

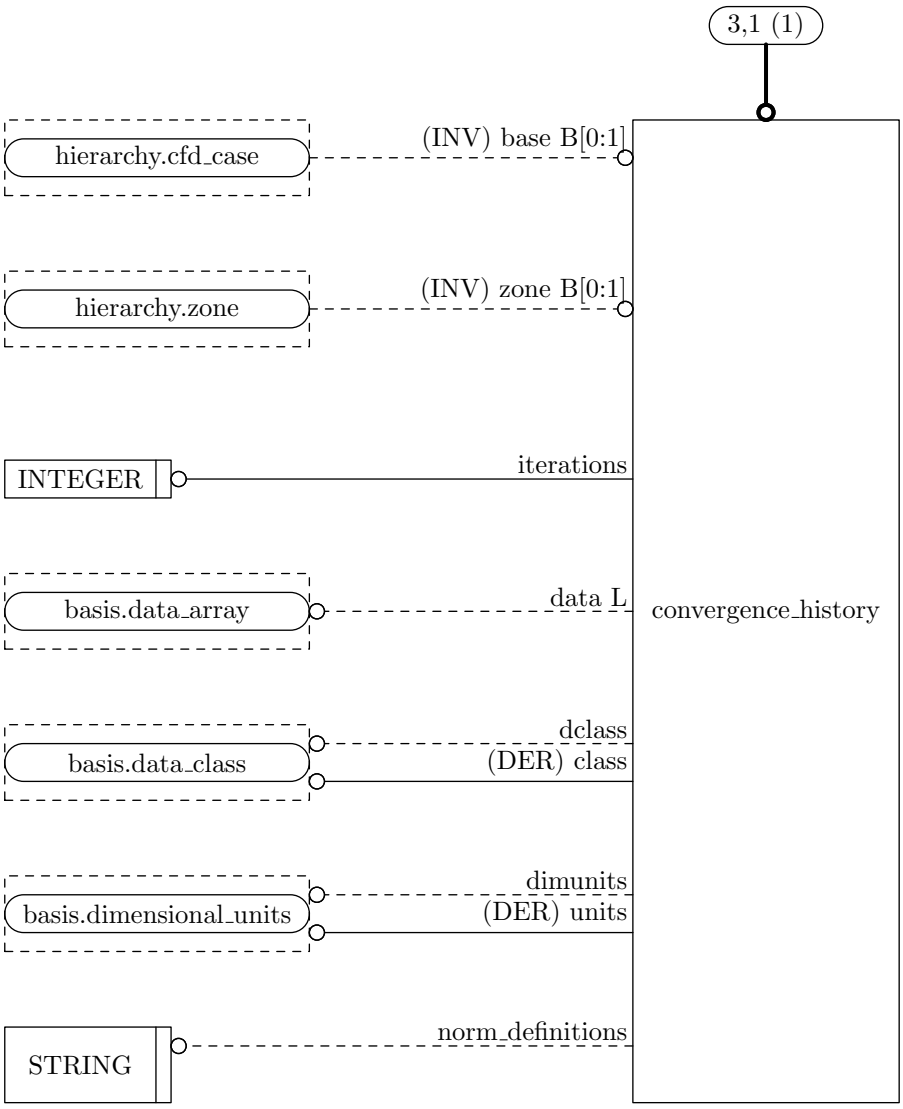


Figure G.40 – Entity level diagram of ARM results schema (page 3 of 4)

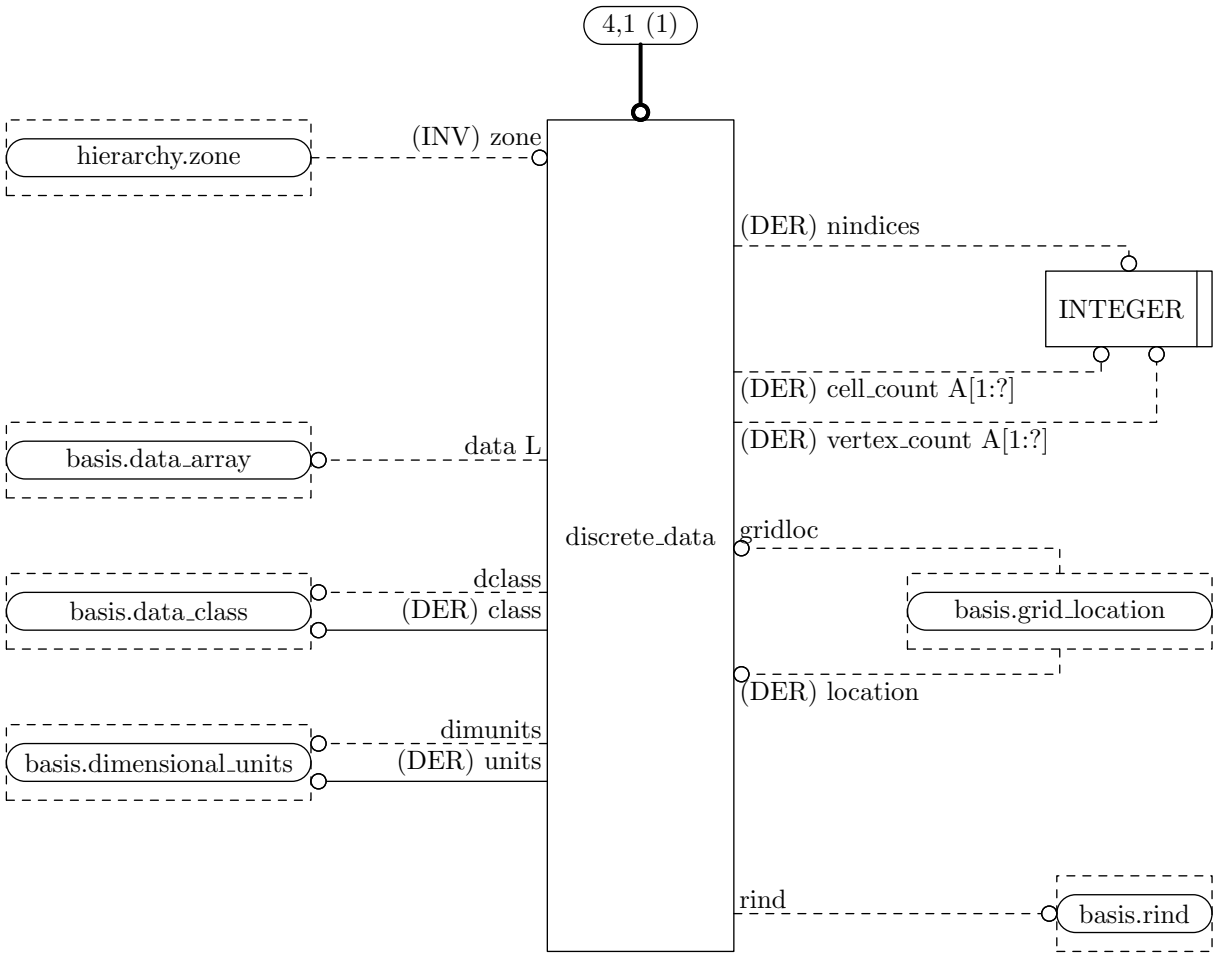


Figure G.41 – Entity level diagram of ARM results schema (page 4 of 4)

and the stress tensor is,

$$\bar{\bar{\tau}}_{i,j} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \lambda \frac{\partial u_k}{\partial x_k},$$

where $(x_1, x_2, x_3) = (x, y, z)$ and $(u_1, u_2, u_3) = (u, v, w)$. The bulk viscosity is usually approximated as $\lambda = -2/3\mu$.

Reynolds averaging of the Navier-Stokes equations introduce Reynolds stresses ($-\rho\overline{u'v'}$, etc.) and turbulent heat flux terms ($-\rho\overline{u'e'}$, etc.), where primed quantities are instantaneous fluctuations and the bar is an averaging operator. These quantities are obtained from auxiliary turbulence closure models. Reynolds-stress models formulate transport equations for the Reynolds stresses directly; whereas, eddy-viscosity models correlate the Reynolds stresses with the mean strain rate,

$$-\overline{u'v'} = \nu_t \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right),$$

where ν_t is the eddy viscosity. The eddy viscosity is either correlated to mean flow quantities

by algebraic models or by auxiliary transport models. An example two-equation turbulence transport model is the k-ϵ model, where transport equations are formulated for the turbulent kinetic energy ($k = \frac{1}{2}(\overline{u'u'} + \overline{v'v'} + \overline{w'w'})$) and turbulent dissipation (ϵ).

Skin friction evaluated at a surface is the dot product of the shear stress tensor with the surface normal:

$$\vec{\tau} = \bar{\bar{\tau}} \cdot \hat{n},$$

Note that skin friction is a vector.

EXPRESS specification:

```
*)
TYPE flow_solution_data_name = EXTENSIBLE ENUMERATION OF ();
END_TYPE;
(*
```

The required identifiers and their meanings are given in Table 6.

G.7.4 results entity definitions

G.7.4.1 result

A **result** represents the concept of a solution to an analysis problem and/or other data resulting from an analysis.

EXPRESS specification:

```
*)
ENTITY result;
  descriptions : OPTIONAL LIST OF textual;
  id           : label;
END_ENTITY;

SUBTYPE_CONSTRAINT sc1_result FOR result;
  ABSTRACT SUPERTYPE;
  ONEOF(flow_solution,
        convergence_history,
        discrete_data);
END_SUBTYPE_CONSTRAINT;
(*
```

Attribute definitions:

descriptions: is annotation;

id: User-specified instance identifier;

G.7.4.2 flow_solution

The flow solution within a given zone is described by the **flow_solution** structure. This structure contains a list of the data arrays of the individual flow solution variables, as well as identifying the grid location of the solution. It also provides a mechanism for identifying rind-point data included within the data arrays.

EXPRESS specification:

```
*)
ENTITY flow_solution
  SUBTYPE OF (result);
  rind        : OPTIONAL rind;
  data        : OPTIONAL LIST OF data_array;
  gridloc     : OPTIONAL grid_location;
  dclass      : OPTIONAL data_class;
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  nindices : INTEGER := zone.nindices;
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  vertex_count : ARRAY [1:nindices] OF INTEGER := zone.vertex_count;
  cell_count   : ARRAY [1:nindices] OF INTEGER := zone.cell_count;
  location    : grid_location := NVL(gridloc, vertex);
INVERSE
  zone : zone FOR solution;
END_ENTITY;
(*
```

Attribute definitions:

rind: is the number of rind planes included in the data.

data: is the data. Each structure in the list contains a single component of the solution vector.

gridloc: is the non-default kind of grid location;

dclass: non-default class of data;

dimunits: non-default system of units;

nindices: The number of indices required to reference a node.

class: is the default class for data in data_arrays.

units: is the description of the system of units.

vertex_count: is the numbers of core vertices in each index direction.

cell_count: is the numbers of core cells in each index direction.

location: specifies the location of the solution data with respect to the grid. All data within a given instance of **flow_solution** resides at the same kind of grid location.

zone: is the zone.

Informal propositions:

ip1: The **rind nindices** shall match the flow solution **nindices**

ip2: A flow solution of an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

ip3: The **data** shall be consistent.

ip4: The grid location shall be either at a vertex or a cell centre.

G.7.4.3 convergence_history

Flow solver convergence history information is described by the **convergence_history** structure.

Measures used to record convergence vary greatly among current flow-solver implementations. Convergence information typically includes global forces, norms of equation residuals, and norms of solution changes.

NOTE 1 Attempts to systematically define a set of convergence measures have been futile. For global parameters, such as forces and moments, clause G.3.3.12 provides a set of standarized data-array identifiers. For equation residuals and solution changes, no such standard list exists. Therefore, **adhoc_data_name** has to be used as the data-array identifier. It is suggested that identifiers for norms of equation residuals begin with RSD, and those for solution changes begin with CHG. For example, **'RSDMassRMS'** could be used for the L_2-norm (RMS) of mass conservation residuals.

EXPRESS specification:

```
*)
ENTITY convergence_history
  SUBTYPE OF (result);
  norm_definitions : OPTIONAL STRING;
  iterations       : INTEGER;
  data             : LIST OF data_array;
  dclass           : OPTIONAL data_class;
  dimunits         : OPTIONAL dimensional_units;
DERIVE
  class        : data_class :=
                       NVL(dclass, inherit_class_from_base_zone(base, zone));
  units : dimensional_units :=
```



```
                    NVL(dimunits, inherit_units_from_base_zone(base, zone));
INVERSE
  base : BAG [0:1] OF cfd_case FOR history;
  zone : BAG [0:1] OF zone FOR history;
END_ENTITY;
(*
```

Attribute definitions:

norm_definitions: is a description of the convergence information recorded as **data**;

iterations: is the number of iterations for which convergence information is recorded;

data: is convergence history data;

dclass: non-default class of data;

dimunits: non-default system of dimensional units;

class: is the class of data;

units: is the system of dimensional units;

base: is the referencing database;

zone: is the referencing zone.

Informal propositions:

ip1: A **convergence_history** instance shall be called by either a **cfd_case** or a **zone**;

ip2: The data arrays shall be consistent.

G.7.4.4 discrete_data

discrete_data provides a description of generic discrete data (i.e., data defined on a computational grid); it is identical to **flow_solution** except for its entity name. This structure can be used to store field data, such as fluxes or equation residuals, that is not typically considered part of the flow solution.

EXPRESS specification:

```
*)
ENTITY discrete_data
  SUBTYPE OF (result);
  gridloc     : OPTIONAl grid_location;
  rind        : OPTIONAL rind;
  data        : LIST OF data_array;
  dclass      : OPTIONAL data_class;
```



```
  dimunits    : OPTIONAL dimensional_units;
DERIVE
  location    : grid_location := NVL(gridloc, vertex);
  class       : data_class := NVL(dclass, zone.class);
  units       : dimensional_units := NVL(dimunits, zone.units);
  nindices   : INTEGER := zone.nindices;
  vertex_count : ARRAY [1:nindices] OF INTEGER := zone.vertex_count;
  cell_count   : ARRAY [1:nindices] OF INTEGER := zone.cell_count;
INVERSE
  zone : zone FOR field_data;
END_ENTITY;
(*
```

Attribute definitions:

gridloc: non-default grid location;

rind: is the Rind planes; if absent then it is equivalent to having an instance of **Rind** whose **RindPlanes** array contains all zeroes.

data: is the data;

dclass: non-default data class;

dimunits: non-default dimensional units;

location: is the location of data with respect to the grid;

class: is the class of data;

units: is the system of dimensional units;

nindices: The number of indices required to reference a node.

vertex_count: is the number of core vertices in each index direction;

cell_count: is the number of core cells in each index direction;

zone: is the calling zone.

Informal propositions:

ip1: The **nindices** of **rind** shall match the discrete data **nindices**.

ip2: Discrete data for an unstructured zone shall not have a value for **rind**, as it is meaningless in this case.

ip3: The data arrays shall be consistent.

ip4: The grid location shall be either a vertex or a cell centre.

ip5: All data contained within this structure shall be defined at the same grid location and have the same amount of rind-point data.

EXPRESS specification:

```
*)
END_SCHEMA; -- end of results
(*
```


Annex H
(informative)
AIM EXPRESS-G

(TBD — FIGURE RANGE) correspond to the AIM EXPRESS expanded listing given in annex A. The diagrams use the EXPRESS-G graphical notation for the EXPRESS language. EXPRESS-G is defined in annex D of ISO 10303-11.

Annex J
(informative)
AIM EXPRESS listing

This annex references a listing of the EXPRESS entity names and corresponding short names as specified in the AIM this part of ISO 10303. It also references a listing of each EXPRESS schema specified in the AIM of this part of ISO 10303, without comments or other explanatory text. These listings are available in computer-interpretable form and can be found at the following URLs:

Short names: `<(TBD---SHORT)>`
EXPRESS: `<(TBD---EXPRESS)>`

If there is difficulty accessing these sites contact ISO Central Secretariat or contact the ISO TC 184/SC4 Secretariat directly at: `sc4sec@cme.nist.gov`.

NOTE The information provided in computer-interpretable form at the above URLs is informative. The information that is contained in the body of this part of ISO 10303 is normative.

Annex K
(informative)
Application protocol usage guide

Annex L
(informative)
Technical discussions

Bibliography

[1] ALLMARA, S. and McCARTHY, D., *CGNS Standard Interface Structures,* 11 August 1999

[2] POIRIER, D., *SIDS additions/modifications to support unstructured meshes and geometry links,* June 1999

[3] CGNS PROJECT GROUP, *The CFD General Notation System: Standard Interface Data Structures,* August 2000

[4] WHITE, F. M., *Viscous Fluid Flow* McGraw-Hill, 1974

[5] *IDEF0 (ICAM Definition Language 0),* Federal Information Processing Standards Publication 183, Integration Definition for Information Modeling (IDEF0), FIPS PUB 183, National Institute for Standards and Technology, December 1993.

Index

adhoc_data_name (type) 113
analysis (entity) 96
angle_units (enumeration) 112
array_of_data (entity) 115
basis (schema) 106
bc (entity) 39, 145
bc_data (entity) 46
bc_data_set (entity) 45
bc_type (select) 21, 140
bc_type_compound (enumeration) 24, 143
bc_type_simple (enumeration) 21, 140
bcdata_class (function) 158
bcdata_dimunits (function) 158
bcdataset_data_array_length (function) 157
cfd_aim (schema) 10
cfd_base (entity) 30
cfd_case (entity) 99
cfd_pdm (entity) 115
cfd_standard_data_name (select) 12
condition (entity) 144
conditions (schema) 129
connectivity (entity) 128
consistent_data_array (function) 63
convergence_history (entity) 58
coordinate_data_name (enumeration) 113
data_array (entity) 119
data_class (enumeration) 111
data_conversion (entity) 116
data_name (select) 112
derive_zone_dimension (function) 105
dimensional_data_array (entity) 120
dimensional_exponents (entity) 117
dimensional_units (entity) 116
discrete_data (entity) 59
domain (schema) 122
element (entity) 104
equation (entity) 175
equations (schema) 162
family (entity) 48, 148
fd_model (entity) 51, 178
field_data_size (function) 67
flow_equation_set (entity) 48, 176
flow_solution (entity) 36, 188
flow_solution_data_name (enumeration) 12, 187
force_moment_data_name (enumeration) 19, 169

gas_model (entity) 51, 179
gas_model_type (enumeration) 25, 170
geometry_reference (entity) 121
governing_equations (entity) 50, 177
governing_equations_type (enumeration) 24, 169
grid_coordinates (entity) 34, 125
grid_data_size (function) 66
grid_location (enumeration) 112
hierarchy (schema) 95
index_list (entity) 118
index_range (entity) 118
inherit_class_from_base_zone (function) 64, 159
inherit_units_from_base_zone (function) 64, 160
inherited_class_for_refstate (function) 65, 160
inherited_units_for_refstate (function) 66, 161
integral_data (entity) 61
label (type) 108
length_units (enumeration) 111
mass_units (enumeration) 111
nondimensional_data_array (entity) 120
nondimensional_data_name (enumeration) 17, 114
product (entity) 101
product_analysis (entity) 100
reference_state (entity) 56
result (entity) 187
results (schema) 182
Riemann_1D_data_name (enumeration) 17, 139
rind (entity) 121
sc1_analysis (subtype constraint) 96
sc1_condition (subtype constraint) 144
sc1_connectivity (subtype constraint) 128
sc1_fd_model (subtype constraint) 51, 178
sc1_result (subtype constraint) 187
sc1_zone (subtype constraint) 31, 101
schdot (function) 62
schname (function) 62
standard_data_name (select) 113
structured_zone (entity) 33, 103
temperature_units (enumeration) 112
textual (type) 109
thermal_conductivity_model (entity) 52, 179
thermal_conductivity_model_type (enumeration) 27, 172
time_units (enumeration) 111
turbulence_closure (entity) 53, 180
turbulence_closure_type (enumeration) 28, 173
turbulence_data_name (enumeration) 15, 163

turbulence_model (entity) 54, 181
turbulence_model_type (enumeration) 29, 174
unstructured_zone (entity) 34, 104
viscosity_model (entity) 55, 182
viscosity_model_type (enumeration) 26, 171
zone (entity) 31, 101
zone_bc (entity) 37, 144
zone_grid_connectivity (entity) 128

